binarly-io/efiXplorer

IDA plugin for UEFI firmware analysis and reverse engineering automation

ref: v6.0

License: GNU General Public License v3.0

Stars: 894

Forks: 105

Top Contributors

yeggor (497)

matrosov (36)

assafcarlsbad (6)

pagabuc (6)

TakahiroHaruyama (5)

isciurus (5)

p41l (2)

RolfRolles (2)

cc-crack (2)

river-li (2)

xorpse (2)

Cr4sh (1)

naconaco (1)

NikolajSchlej (1)

Chapter 0.0.0

root

README.md

efiXplorer - IDA plugin for UEFI firmware analysis and reverse engineering automation

Supported versions of Hex-Rays products: everytime we focus on last versions of IDA and Decompiler because we try to use most recent features from new SDK releases. That means we tested only on recent versions of Hex-Rays products and do not guarantee stable work on previous generations.

Why not IDApython: all code developed in C++ because it's a more stable and performant way to support a complex plugin and get full power of most recent SDK's features.

Supported Platforms: Windows, Linux and OSX.

efiXplorer core features

efiXloader description

Build instructions and Installation

Publications

- efiXplorer: Hunting for UEFI Firmware Vulnerabilities at Scale with Automated Static Analysis
- Static analysis-based recovery of service function calls in UEFI firmware
- How efiXplorer helping to solve challenges in reverse engineering of UEFI firmware

References

- https://github.com/LongSoft/UEFITool
- https://github.com/yeggor/uefi_retool
- https://github.com/gdbinit/EFISwissKnife
- https://github.com/snare/ida-efiutils
- https://github.com/al3xtjames/ghidra-firmware-utils
- https://github.com/DSecurity/efiSeek
- https://github.com/p-state/ida-efitools2
- https://github.com/zznop/bn-uefi-helper

LICENSE

the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically

linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

  The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

  The Corresponding Source for a work in source code form is that same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

  When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This

alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and

protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

    e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

  All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you

must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment

to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<https://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<https://www.gnu.org/licenses/why-not-lgpl.html>.
}
```

build.py

```
#!/usr/bin/env python3

import os
import subprocess

import click


@click.group()
def cli():
    pass


@click.command()
@click.option(
    "--batch",
    "batch",
    type=bool,
    default=False,
    help="set to True if the plugin will be used in batch mode",
)
@click.option(
    "--hexrays_sdk",
    "hexrays_sdk",
    type=str,
    default=str(),
    help="path to hexrays_sdk directory",
)
@click.argument("idasdk")
def build_plugin(idasdk: str, hexrays_sdk: str, batch: bool):
    """Build efiXplorer plugin"""

    os.chdir("efiXplorer")

    if not os.path.isdir("build"):
        os.mkdir("build")
```

```
    os.chdir("build")

    command = ["cmake", "..", f"-DIdaSdk_ROOT_DIR={idasdk}"]
    if batch:
        command.append("-DBATCH=1")
    if hexrays_sdk:
        print("[INFO] HexRays analysis will be enabled")
        command.append(f"-DHexRaysSdk_ROOT_DIR={hexrays_sdk}")
    subprocess.call(command)
    subprocess.call(["cmake", "--build", ".", "--config", "Release", "--parallel"])


@click.command()
@click.argument("idasdk")
def build_loader(idasdk: str):
    """Build efiXloader"""

    os.chdir("efiXloader")

    if not os.path.isdir("build"):
        os.mkdir("build")

    os.chdir("build")

    command = ["cmake", "..", f"-DIdaSdk_ROOT_DIR={idasdk}"]
    subprocess.call(command)
    subprocess.call(["cmake", "--build", ".", "--config", "Release", "--parallel"])


cli.add_command(build_plugin)
cli.add_command(build_loader)


if __name__ == "__main__":
    cli()
}
```

Chapter 1.0.0

efiXloader

efiXloader/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.7)

project(efiXloader)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

if(APPLE)
  # to build Mach-O universal binaries with 2 architectures
```

```
  set(CMAKE_CXX_FLAGS ${CMAKE_CXX_FLAGS} "-fPIC -arch x86_64 -arch arm64")
  set(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-fPIC -arch x86_64 -arch arm64")
else()
  set(CMAKE_CXX_FLAGS ${CMAKE_CXX_FLAGS} "-fPIC")
endif()

list(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/../cmake)

find_package(IdaSdk REQUIRED)

include_directories(${PROJECT_SOURCE_DIR}/../efiXplorer/3rd/nlohmann_json)

file(
  GLOB
  uefitool_src
  "3rd/uefitool/common/*.h"
  "3rd/uefitool/common/*.c"
  "3rd/uefitool/common/*.cpp"
  "3rd/uefitool/common/LZMA/*.cpp"
  "3rd/uefitool/common/LZMA/*.c"
  "3rd/uefitool/common/LZMA/*.h"
  "3rd/uefitool/common/LZMA/SDK/C/*.c"
  "3rd/uefitool/common/LZMA/SDK/C/*.cpp"
  "3rd/uefitool/common/LZMA/SDK/C/*.h"
  "3rd/uefitool/common/Tiano/*.c"
  "3rd/uefitool/common/Tiano/*.cpp"
  "3rd/uefitool/common/Tiano/*.h"
  "3rd/uefitool/common/bstrlib/*.c"
  "3rd/uefitool/common/bstrlib/*.cpp"
  "3rd/uefitool/common/bstrlib/*.h"
  "3rd/uefitool/common/digest/sha1.c"
  "3rd/uefitool/common/digest/sha256.c"
  "3rd/uefitool/common/digest/sha512.c"
  "3rd/uefitool/common/digest/sm3.c"
  "3rd/uefitool/common/zlib/*.c"
  "3rd/uefitool/common/zlib/*.cpp"
  "3rd/uefitool/common/zlib/*.h"
  "3rd/uefitool/version.h"
  "3rd/uefitool/ffsdumper.cpp"
  "3rd/uefitool/ffsdumper.h"
  "3rd/uefitool/uefidump.cpp"
  "3rd/uefitool/uefidump.h")

# efiLoader sources
file(GLOB efiloader_src "*.h" "*.c" "*.cpp")

add_ida_loader(efiXloader NOEA32 ${PROJECT_SOURCE_DIR}/efiLoader.cpp)

set_ida_target_properties(efiXloader PROPERTIES CXX_STANDARD 17)
ida_target_include_directories(efiXloader PRIVATE ${IdaSdk_INCLUDE_DIRS})

add_ida_library(efiXloader_lib ${efiloader_src} ${uefitool_src} uefitool.cpp
                uefitool.h)
ida_target_link_libraries(efiXloader efiXloader_lib)
}
```

efiXloader/efiLoader.cpp

```
/*
 * efiXloader
```

```
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiLoader.cpp
 */

#include "efiLoader.h"
#include "uefitool.h"
#include "utils.h"

#define USE_UEFITOOL_PARSER

bool first_uefi_image = true;

//-------------------------
// IDA wrappers
//-------------------------

void idaapi load_binary(const char *fname) {
    load_info_t *ld = NULL;
    linput_t *li = NULL;
    ushort nflags = NEF_SEGS | NEF_RSCS | NEF_NAME | NEF_IMPS | NEF_LALL | NEF_FLAT;
    if (first_uefi_image) {
        nflags |= NEF_FIRST;
    }
    first_uefi_image = false;
    // linput
    li = open_linput(fname, false);
    if (li == NULL) {
        error("failed to process input source: %s", fname);
    }
    // get loaders
    ld = build_loaders_list(li, fname);
    msg("[efiXloader] using %s to load %s\n", ld->dllname.c_str(), fname);
    // load EFI binary into database
    if ((load_nonbinary_file(fname, li, ".", nflags, ld))) {
        msg("[efiXloader] successfully loaded %s\n", fname);
    } else {
        loader_failure("[efiXloader] 'load_nonbinary_file' failed");
    }
    close_linput(li);
    free_loaders_list(ld);
    return;
}

void idaapi wait(void) {
    while (!auto_is_ok()) {
        auto_wait();
    }
}

//-------------------------
```

```
// IDA analyzing
//------------------------

void inline idaapi reanalyze_all(void) {
    plan_range(inf_get_min_ea(), inf_get_max_ea());
    auto_wait();
    auto_make_proc(inf_get_min_ea());
}

void efi_til_init(const char *til_name) {
    qstring err;
    til_t *res;
    res = load_til(til_name, &err);
    if (!res) {
        loader_failure("failed to load %s", til_name);
    } else {
        msg("[efiXloader] lib %s is ready\n", til_name);
    }
}

//------------------------
// IDA loader
//------------------------

static int idaapi accept_file(qstring *fileformatname, qstring *processor,
linput_t *li,
                              const char *filename) {
    efiloader::Utils utils;
    bytevec_t data;
    data.resize(qlsize(li));
    qlseek(li, 0);
    qlread(li, data.begin(), qlsize(li));
    *fileformatname = "UEFI firmware image";
    return utils.find_vol_test(data) != std::string::npos;
}

void idaapi load_file(linput_t *li, ushort neflag, const char *fileformatname) {
    bool ok = true;
    bool is_pe;
    Ui ui;
    bytevec_t data;
    data.resize(qlsize(li));
    qlread(li, data.begin(), qlsize(li));
    efiloader::Uefitool uefiParser(data);
    if (uefiParser.messages_occurs()) {
        uefiParser.show_messages();
    }
    uefiParser.dump();
    uefiParser.dump_jsons();
    msg("[efiXloader] machine type: %04x\n", uefiParser.machine_type);
    efiloader::PeManager peManager(uefiParser.machine_type);
    add_til("uefi.til", ADDTIL_DEFAULT);
    add_til("uefi64.til", ADDTIL_DEFAULT);
    qstring err;
    const til_t *idati = get_idati();
    if (!idati) {
        loader_failure("failed to load IDA types");
    } else {
        msg("[efiXloader] loaded IDA types\n");
    }
    tid_t struct_err = import_type(idati, -1, "EFI_GUID");
    if (struct_err == BADNODE) {
        loader_failure("failed to import \"EFI_GUID\"");
    }
```

```
    msg("processing UEFI binaries:\n");
    if (uefiParser.files.size()) {
        for (int i = 0; i < uefiParser.files.size(); i++) {
            if (uefiParser.files[i]->is_te) {
                continue;
            }
            auto inf = open_linput(uefiParser.files[i]->dump_name.c_str(), false);
            if (!inf) {
                msg("Unable to open file %s\n", uefiParser.files[i]-
>dump_name.c_str());
                continue;
            }
            peManager.process(inf, uefiParser.files[i]->dump_name.c_str(), i);
        }
    } else {
        msg("[efiXloader] Can not parse input firmware\n");
    }

    plugin_t *findpat = find_plugin("patfind", true);
    if (findpat) {
        msg("Running the Find functions plugin\n");
        run_plugin(findpat, 0);
    }
}

static int idaapi move_segm(ea_t from, ea_t to, asize_t, const char *) { return 1; }

//--------------------------------------------------------------------------
//
//      LOADER DESCRIPTION BLOCK
//
//--------------------------------------------------------------------------
loader_t LDSC = {
    IDP_INTERFACE_VERSION,
    // loader flags
    0,
    // check input file format. if recognized, then return 1
    // and fill 'fileformatname'.
    // otherwise return 0
    accept_file,
    // load file into the database.
    load_file,
    // create output file from the database.
    // this function may be absent.
    NULL,
    // take care of a moved segment (fix up relocations, for example)
    NULL,
    NULL,
};
}
```

efiXloader/efiLoader.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
```

```
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiLoader.h
 */

#ifndef EFILOADER_EFILOADER_H
#define EFILOADER_EFILOADER_H

#include "ida_core.h"
#include "pe.h"
#include "pe_manager.h"
#include "uefitool.h"

extern loader_t LDSC;

//-----------------------
// definitions
//-----------------------

void idaapi load_binary(const char *fname);
void idaapi close_and_save_db(const char *fname);
void idaapi reanalyze_all(void);
void idaapi wait(void);
void idaapi idb_to_asm(const char *fname);
void idaapi clean_db(void);

void idaapi efi_til_init();

// UI

class Ui {
  public:
    Ui() { ; }
    int ask_for_single_image();
};

class driver_chooser_t : public chooser_t {
  protected:
    static const int widths_drivers[];
    static const char *const drivers_headers[];

  public:
    /* remember the addresses in this qvector */
    qvector<qstring> drivers_names;

    /* this object must be allocated using `new` */
    driver_chooser_t(const char *title, bool ok, std::vector<efiloader::File
*> drivers);

    /* function that is used to decide whether a new chooser should be opened or
     * we can use the existing one. The contents of the window are completely
     * determined by its title */
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }
```

```
    /* function that returns number of lines in the list */
    virtual size_t idaapi get_count() const { return drivers_names.size(); }

    /* function that generates the list line */
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;

    /* function that is called when the user hits Enter */
    virtual cbret_t idaapi enter(size_t n) {
        if (n < drivers_names.size()) {
            // jumpto(list[n]);
        }
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<efiloader::File *> files) {
        size_t n = 0;
        for (auto file : files) {
            drivers_names.push_back(qstring(file->qname));
            n++;
        }
        ok = true;
    };
};

#endif // EFILOADER_EFILOADER_H
}
```

efiXloader/ida_core.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * ida_core.h
 */

#ifndef EFILOADER_IDA_CORE_H
#define EFILOADER_IDA_CORE_H

#define USE_STANDARD_FILE_FUNCTIONS 1

#include <auto.hpp>
#include <bytes.hpp>
```

```
#include <diskio.hpp>
#include <entry.hpp>
#include <fixup.hpp>
#include <fpro.h>
#include <ida.hpp>
#include <idp.hpp>
#include <kernwin.hpp>
#include <loader.hpp>
#include <name.hpp>
#include <offset.hpp>
#include <pro.h>
#include <segment.hpp>
#include <segregs.hpp>

//--------------------------------

#define CLASS_CODE "CODE"
#define NAME_CODE ".text"
#define CLASS_DATA "DATA"
#define CLASS_CONST "CONST"
#define NAME_DATA ".data"
#define CLASS_BSS "BSS"
#define NAME_BSS ".bss"
#define NAME_EXTERN "extern"
#define NAME_COMMON "common"
#define NAME_ABS "abs"
#define NAME_UNDEF "UNDEF"
#define CLASS_STACK "STACK"
#define CLASS_RES16 "RESOURCE"
#define LDR_NODE "$ IDALDR node for ids loading $"
#define LDR_INFO_NODE "$ IDALDR node for unload $"

//-------------------------------------------------------------------------
template <class T>
bool _validate_array_count(linput_t *li, T *p_cnt, size_t elsize,
                           int64 current_offset = -1, int64 max_offset = -1) {
    if (current_offset == -1)
        current_offset = qltell(li);
    if (max_offset == -1)
        max_offset = qlsize(li);
    int64 rest = max_offset - current_offset;
    T cnt = *p_cnt;
    if (current_offset >= 0 && rest >= 0) {
#ifndef __X86__
        typedef size_t biggest_t;
#else
        typedef ea_t biggest_t;
#endif
        if (is_mul_ok<biggest_t>(elsize, cnt)) {
            biggest_t needed = elsize * cnt;
#ifdef __X86__
            if (needed == size_t(needed))
#endif
                if (rest >= needed)
                    return true; // all ok
        }
        cnt = rest / elsize;
    } else {
        cnt = 0;
    }
    *p_cnt = cnt;
    return false;
}
```

```
//-------------------------------------------------------------------------
// Validate a counter taken from the input file. If there are not enough bytes
// in the input file, ask the user if we may continue and fix the counter.
template <class T>
void validate_array_count(linput_t *li, T *p_cnt, size_t elsize, const
char *counter_name,
                          int64 curoff = -1, int64 maxoff = -1) {
    T old = *p_cnt;
    if (!_validate_array_count(li, p_cnt, elsize, curoff, maxoff)) {
        static const char *const format =
            "AUTOHIDE SESSION\n"
            "HIDECANCEL\n"
            "%s %" FMT_64 "u is incorrect, maximum possible value is %" FMT_64 "u%s";
#ifndef __KERNEL__
        if (ask_yn(ASKBTN_YES, format, counter_name, uint64(old), uint64(*p_cnt),
                   ". Do you want to continue with the new value?") != ASKBTN_YES) {
            loader_failure(NULL);
        }
#else
        warning(format, counter_name, uint64(old), uint64(*p_cnt), "");
#endif
    }
}

//-------------------------------------------------------------------------
// Validate a counter taken from the input file. If there are not enough bytes
// in the input file, die.
template <class T>
void validate_array_count_or_die(linput_t *li, T cnt, size_t elsize,
                                 const char *counter_name, int64 curoff = -1,
                                 int64 maxoff = -1) {
    if (!_validate_array_count(li, &cnt, elsize, curoff, maxoff)) {
        static const char *const format =
            "%s is incorrect, maximum possible value is %u%s";
#ifndef __KERNEL__
        loader_failure(format, counter_name, uint(cnt), "");
#else
        error(format, counter_name, uint(cnt), "");
#endif
    }
}

//----------------------------------
inline uchar readchar(linput_t *li) {
    uchar x;
    lread(li, &x, sizeof(x));
    return x;
}

//----------------------------------
inline uint16 readshort(linput_t *li) {
    uint16 x;
    lread(li, &x, sizeof(x));
    return x;
}

//----------------------------------
inline uint32 readlong(linput_t *li) {
    uint32 x;
    lread(li, &x, sizeof(x));
    return x;
}

inline uint32 mf_readlong(linput_t *li) { return swap32(readlong(li)); }
```

```
inline uint16 mf_readshort(linput_t *li) { return swap16(readshort(li)); }

#endif // EFILOADER_IDA_CORE_H
}
```

efiXloader/pe.cpp

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * pe.cpp
 */

#include "pe.h"

#include <bytes.hpp>
#include <fixup.hpp>
#include <idp.hpp>
#include <map>
#include <pro.h>
#include <segregs.hpp>

#define DEBUG

#define DIRECTORIES_MAX_ID 15

std::map<uint32_t, const char *> DIRECTORIES = {
    {0, "Export Directory"},
    {1, "Import Directory"},
    {2, "Resource Directory"},
    {3, "Exception Directory"},
    {4, "Security Directory"},
    {5, "Base Relocation Table"},
    {6, "Debug Directory"},
    {7, "Architecture Specific Data"},
    {8, "RVA of GP"},
    {9, "TLS Directory"},
    {10, "Load Configuration Directory"},
    {11, "Bound Import Directory in headers"},
    {12, "Import Address Table"},
    {13, "Delay Load Import Descriptors"},
    {14, "COM Runtime descriptor"},
    {15, "Image data directory"},
};

//
```

```
// efiLoader core routines
//

qoff64_t efiloader::PE::head_start() {
    qoff64_t off = 0;
    qlseek(li, 0x3c);
    off = readshort(li);
    reset();
    return off;
}

//
// Functions to extract PE features
//

bool efiloader::PE::good() {
    uint16_t mz = 0;
    qlread(li, &mz, sizeof(uint16_t));
    if (mz != MZ_SIGN) {
        return false;
    }
    _pe_header_off = head_start();
    if (is_p32_plus()) {
        _bits = 64;
    } else if (!is_p32()) {
        loader_failure("[efiXloader] failed to guess PE bitness");
    } else {
        _bits = 32;
    }
    return is_pe();
}

bool efiloader::PE::is_p32() {
    uint16_t magic = 0;
    qlseek(li, _pe_header_off + sizeof(uint32_t));
    qlread(li, &magic, sizeof(uint16_t));
    reset();
    return magic == I386;
}

bool efiloader::PE::is_p32_plus() {
    uint16_t magic = 0;
    qlseek(li, _pe_header_off + sizeof(uint32_t));
    qlread(li, &magic, sizeof(uint16_t));
    reset();
    return (magic == AMD64 || magic == AARCH64);
}

bool efiloader::PE::is_pe() {
    uint16_t pe_sign = 0;
    qlseek(li, _pe_header_off);
    pe_sign = readshort(li);
    reset();
    return pe_sign == PE_SIGN;
}

uint16_t efiloader::PE::arch() {
    qlseek(li, head_off + sizeof(uint32_t));
    qlread(li, &pe, sizeof(peheader_t));
    return pe.machine;
}

bool efiloader::PE::process() {
    preprocess();
```

```
    *pe_base = image_base + image_size;
    return true;
}

const char *efiloader::PE::_machine_name() {
    switch (pe64.machine) {
    case PECPU_80386:
        return "80386";
    case PECPU_80486:
        return "80486";
    case PECPU_80586:
        return "80586";
    case PECPU_SH3:
        return "SH3";
    case PECPU_SH3DSP:
        return "SH3DSP";
    case PECPU_SH3E:
        return "SH3E";
    case PECPU_SH4:
        return "SH4";
    case PECPU_SH5:
        return "SH5";
    case PECPU_ARM:
        return "ARM";
    case PECPU_ARMI:
        return "ARMI";
    case PECPU_ARMV7:
        return "ARMv7";
    case PECPU_EPOC:
        return "ARM EPOC";
    case PECPU_PPC:
        return "PPC";
    case PECPU_PPCFP:
        return "PPC FP";
    case PECPU_PPCBE:
        return "PPC BE";
    case PECPU_IA64:
        return "IA64";
    case PECPU_R3000:
        return "MIPS R3000";
    case PECPU_R4000:
        return "MIPS R4000";
    case PECPU_R6000:
        return "MIPS R6000";
    case PECPU_R10000:
        return "MIPS R10000";
    case PECPU_MIPS16:
        return "MIPS16";
    case PECPU_WCEMIPSV2:
        return "MIPS WCEv2";
    case PECPU_ALPHA:
        return "ALPHA";
    case PECPU_ALPHA64:
        return "ALPHA 64";
    case PECPU_AMD64:
        return "AMD64";
    case PECPU_ARM64:
        return "ARM64";
    case PECPU_M68K:
        return "M68K";
    case PECPU_MIPSFPU:
        return "MIPS FPU";
    case PECPU_MIPSFPU16:
        return "MIPS16 FPU";
```

```
    case PECPU_EBC:
        return "EFI Bytecode";
    case PECPU_AM33:
        return "AM33";
    case PECPU_M32R:
        return "M32R";
    case PECPU_CEF:
        return "CEF";
    case PECPU_CEE:
        return "CEE";
    case PECPU_TRICORE:
        return "TRICORE";
    }
    return NULL;
}

//
// Functions processing
//

void efiloader::PE::make_entry(ea_t ea) {
    char func_name[MAXNAMESIZE] = {0};
    ea_t new_ea = image_base + ea;
    qsnprintf(func_name, sizeof(func_name), "%s_entry_%08X", _image_name.c_str(),
              calc_file_crc32(li));
    add_entry(new_ea, new_ea, func_name, true);
    memset(func_name, 0, sizeof(func_name));
}

//
// PE image pre-processing
//

inline size_t efiloader::PE::make_named_word(ea_t ea, const char *name, const
char *extra,
                                             size_t count) {
    if (extra) {
        add_extra_cmt(ea, true, "%s", extra);
    }
    create_word(ea, 2 * count);
    set_cmt(ea, name, 0);
    op_hex(ea, 0);
    return 2 * count;
}

inline size_t efiloader::PE::make_named_dword(ea_t ea, const char *name,
                                              const char *extra, size_t count) {
    if (extra) {
        add_extra_cmt(ea, true, "%s", extra);
    }
    create_word(ea, 4 * count);
    set_cmt(ea, name, 0);
    op_hex(ea, 0);
    return 4 * count;
}

inline size_t efiloader::PE::make_named_qword(ea_t ea, const char *name,
                                              const char *extra, size_t count) {
    if (extra) {
        add_extra_cmt(ea, true, "%s", extra);
    }
    create_word(ea, 8 * count);
    set_cmt(ea, name, 0);
    op_hex(ea, 0);
```

```
    return 8 * count;
}

//
// Segments processing
//

segment_t *efiloader::PE::make_head_segment(ea_t start, ea_t end,
                                            const char *section_name) {
    segment_t *seg = new segment_t;
    seg->bitness = 2;
    seg->perm = SEG_DATA;
    seg->sel = allocate_selector(0x0);
    seg->start_ea = start;
    seg->end_ea = end;
    add_segm_ex(seg, section_name, "DATA", ADDSEG_NOAA | ADDSEG_NOSREG);
    return seg;
}

segment_t *efiloader::PE::make_generic_segment(ea_t seg_ea, ea_t seg_ea_end,
                                               const char *section_name, uint32_t
flags) {
    segment_t *generic_segm = new segment_t;
    generic_segm->sel = allocate_selector(0x0);
    generic_segm->start_ea = seg_ea;
    generic_segm->end_ea = seg_ea_end;
    generic_segm->bitness = 2;
    generic_segm->perm = SEGPERM_READ;
    if (flags & PEST_EXEC)
        generic_segm->perm |= SEGPERM_EXEC;
    if (flags & PEST_WRITE)
        generic_segm->perm |= SEGPERM_WRITE;

    qstring name(section_name);

    if (name.find('.') == qstring::npos) {
        name += qstring(".unkn");
    }

    if (flags & PEST_EXEC) {
        generic_segm->type = SEG_CODE;
        add_segm_ex(generic_segm, name.c_str(), "CODE", ADDSEG_NOAA);
    } else {
        generic_segm->type = SEG_DATA;
        add_segm_ex(generic_segm, name.c_str(), "DATA", ADDSEG_NOAA);
    }

    if (name == ".text") {
        code_segm_name.insert(name);
    } else if (name == ".data") {
        data_segm_name.insert(name);
        data_segment_sel = get_segm_by_name(data_segm_name.c_str())->sel;
    }

    secs_names.push_back(name);
    return generic_segm;
}

int efiloader::PE::preprocess_sections() {
    qlseek(li, _pe_header_off);
    qlread(li, &pe, sizeof(peheader_t));

    // x86
    number_of_sections = pe.nobjs;
```

```
    int section_headers_offset = pe.first_section_pos(_pe_header_off);
    headers_size = pe.allhdrsize;

    if (pe.machine == PECPU_AMD64 || pe.machine == PECPU_ARM64) { // AMD64/AARCH64
        qlseek(li, _pe_header_off);
        qlread(li, &pe64, sizeof(peheader64_t));
        number_of_sections = pe64.nobjs;
        section_headers_offset = pe64.first_section_pos(_pe_header_off);
        headers_size = pe64.allhdrsize;
    }

    if (!headers_size) {
        return -1;
    }

    _sec_headers.resize(number_of_sections);
    qlseek(li, section_headers_offset);
    for (int i = 0; i < number_of_sections; i++) {
        qlread(li, &_sec_headers[i], sizeof(pesection_t));
    }

    return 0;
}

ea_t efiloader::PE::process_section_entry(ea_t next_ea) {
    create_strlit(next_ea, 8, STRTYPE_C);
    set_cmt(next_ea, "Name", 0);
    op_hex(next_ea, 0);
    size_t segm_name_len = get_max_strlit_length(next_ea, STRTYPE_C);

    if (segm_name_len) {
        get_strlit_contents(&segm_names.push_back(), next_ea,
segm_name_len, STRTYPE_C);
    } else {
        // if the segm_name_len is 0, it will trigger a crash on segm_names.pop_back()
        // later.
        segm_names.push_back("UNKNOWN");
    }

    next_ea += 8;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Virtual size", 0);
    op_hex(next_ea, 0);
    segm_sizes.push_back(get_dword(next_ea));
    next_ea += 4;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Virtual address", 0);
    op_hex(next_ea, 0);
    segm_entries.push_back(get_dword(next_ea));
    next_ea += 4;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Size of raw data", 0);
    op_hex(next_ea, 0);
    segm_raw_sizes.push_back(get_dword(next_ea));
    next_ea += 4;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Pointer to raw data", 0);
    op_hex(next_ea, 0);
    next_ea += 4;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Pointer to relocations", 0);
    op_hex(next_ea, 0);
    next_ea += 4;
    create_dword(next_ea, 4);
```

```
    set_cmt(next_ea, "Pointer to line numbers", 0);
    op_hex(next_ea, 0);
    next_ea += 4;
    create_word(next_ea, 2);
    set_cmt(next_ea, "Number of relocations", 0);
    op_hex(next_ea, 0);
    next_ea += 2;
    create_word(next_ea, 2);
    set_cmt(next_ea, "Number of linenumbers", 0);
    op_hex(next_ea, 0);
    next_ea += 2;
    create_dword(next_ea, 4);
    set_cmt(next_ea, "Characteristics", 0);
    op_hex(next_ea, 0);
    uint32_t section_characteristics = get_dword(next_ea);
    next_ea += 4;

    qstring section_name = qstring(_image_name.c_str());
    section_name += qstring("_") + qstring(segm_names[0].c_str());

    ea_t seg_ea = image_base + segm_entries[0];
    ea_t seg_ea_end = seg_ea + segm_raw_sizes[0];
    msg("[efiXloader]\tprocessing: %s\n", segm_names[0].c_str());

    segments.push_back(make_generic_segment(seg_ea, seg_ea_end, section_name.c_str(),
                                            section_characteristics));
    segm_names.pop_back();
    segm_sizes.pop_back();
    segm_raw_sizes.pop_back();
    segm_entries.pop_back();
    return next_ea;
}

void efiloader::PE::setup_ds_selector() {
    for (; !secs_names.empty(); secs_names.pop_back()) {
        msg("[efiXloader]\tsetting DS ( 0x%016llX ) for %s segment\n",
            static_cast<uint64_t>(data_segment_sel),
            secs_names[secs_names.size() - 1].c_str());
        segment_t *seg = get_segm_by_name(secs_names[secs_names.size() - 1].c_str());
        set_default_sreg_value(seg, str2reg("DS"), data_segment_sel);
    }
}

//
// PE core processing
//

void efiloader::PE::preprocess() {
    char seg_name[MAXNAMESIZE] = {0};
    char seg_header_name[MAXNAMESIZE] = {0};
    char image_base_name[MAXNAMESIZE] = {0};
    char section_name[MAXNAMESIZE] = {0};
    ea_t next_ea = 0;
    ea_t ea = align_up(*pe_base, PAGE_SIZE);
    image_base = ea;
    ea_t start = ea;
    ea_t end = ea + qlsize(li);
    qsnprintf(seg_name, sizeof(seg_name), "%s_%08X", _image_name.c_str(),
              calc_file_crc32(li));
    qsnprintf(seg_header_name, sizeof(seg_header_name),
"%s_HEADER", _image_name.c_str());
    qsnprintf(image_base_name, sizeof(image_base_name), "%s_IMAGE_BASE",
              _image_name.c_str());
```

```
    if (preprocess_sections() == -1) {
        msg("[efiXloader]\tcannot load %s\n", _image_name.c_str());
        image_base = 0;
        image_size = 0;
        return;
    }
    push_to_idb(start, end);
    segments.push_back(
        make_head_segment(image_base, image_base + headers_size, seg_header_name));
    secs_names.push_back(qstring(seg_header_name));
    create_word(ea, 2);
    set_cmt(ea, "PE magic number", 0);
    op_hex(ea, 0);
    create_word(ea + 2, 2);
    set_cmt(ea + 2, "Bytes on last page of file", 0);
    op_hex(ea + 2, 0);
    create_word(ea + 4, 2);
    set_cmt(ea + 4, "Pages in file", 0);
    op_hex(ea + 4, 0);
    create_word(ea + 6, 2);
    set_cmt(ea + 6, "Relocations", 0);
    op_hex(ea + 6, 0);
    create_word(ea + 8, 2);
    set_cmt(ea + 8, "Size of header in paragraphs", 0);
    op_hex(ea + 8, 0);
    create_word(ea + 10, 2);
    set_cmt(ea + 10, "Minimum extra paragraphs needed", 0);
    op_hex(ea + 10, 0);
    create_word(ea + 10, 2);
    set_cmt(ea + 12, "Maximum extra paragraphs needed", 0);
    op_hex(ea + 12, 0);
    create_word(ea + 14, 2);
    set_cmt(ea + 14, "Initial (relative) SS value", 0);
    op_hex(ea + 12, 0);
    create_word(ea + 16, 2);
    set_cmt(ea + 16, "Initial SP value", 0);
    op_hex(ea + 16, 0);
    create_word(ea + 18, 2);
    set_cmt(ea + 18, "Checksum", 0);
    op_hex(ea + 18, 0);
    create_word(ea + 20, 2);
    set_cmt(ea + 20, "Initial IP value", 0);
    op_hex(ea + 20, 0);
    create_word(ea + 22, 2);
    set_cmt(ea + 22, "Initial (relative) CS value", 0);
    op_hex(ea + 22, 0);
    create_word(ea + 24, 2);
    set_cmt(ea + 24, "File address of relocation table", 0);
    op_hex(ea + 24, 0);
    op_offset(ea + 24, 0, REF_OFF64, BADADDR, image_base);
    create_word(ea + 26, 2);
    set_cmt(ea + 26, "Overlay number", 0);
    op_hex(ea + 26, 0);
    create_word(ea + 28, 8);
    set_cmt(ea + 28, "Reserved words", 0);
    op_hex(ea + 28, 0);
    create_word(ea + 36, 2);
    set_cmt(ea + 36, "OEM identifier (for e_oeminfo)", 0);
    op_hex(ea + 36, 0);
    create_word(ea + 38, 2);
    set_cmt(ea + 38, "OEM information; e_oemid specific", 0);
    op_hex(ea + 38, 0);
    create_word(ea + 40, 2);
    set_cmt(ea + 40, "Reserved words", 0);
```

```
op_hex(ea + 38, 0);
ea_t old_ea = ea;
ea = ea + 0x3c;
create_dword(ea, 4);
if (is_loaded(ea) && get_dword(ea)) {
    msg("[efiXloader] making relative offset: 0x%016llX\n",
        static_cast<uint64_t>(ea));
    op_plain_offset(ea, 0, *pe_base);
}
set_cmt(ea, "File address of new exe header", 0);
uint32_t nt_headers_off = get_dword(ea);
create_byte(old_ea + 0x40, nt_headers_off - 0x40);
set_cmt(old_ea + 0x40, "DOS Stub code", 0);
ea_t nt_headers_ea = old_ea + nt_headers_off;
add_extra_cmt(nt_headers_ea, 1, "IMAGE_NT_HEADERS");
switch (get_word(old_ea)) {
case 0x20B:
    del_items(nt_headers_ea, 3, 0x108);
    break;
default:
    del_items(nt_headers_ea, 3, 0xf8);
    break;
}
create_dword(nt_headers_ea, 4);
set_cmt(nt_headers_ea, "Signature", 0);
op_hex(nt_headers_ea, 0);
ea_t image_file_header_ea = nt_headers_ea + 4;
add_extra_cmt(image_file_header_ea, 1, "IMAGE_FILE_HEADER");
create_word(image_file_header_ea, 2);
set_cmt(image_file_header_ea, "Machine", 0);
op_hex(image_file_header_ea, 0);
image_file_header_ea += 2;
create_word(image_file_header_ea, 2);
set_cmt(image_file_header_ea, "Number of sections", 0);
number_of_sections = get_word(image_file_header_ea);
op_hex(image_file_header_ea, 0);
ea_t timestamp_ea = image_file_header_ea + 2;
create_dword(timestamp_ea, 4);
set_cmt(timestamp_ea, "Time stamp", 0);
ea_t pointer_to_symbol_table = timestamp_ea + 4;
create_dword(pointer_to_symbol_table, 4);
set_cmt(pointer_to_symbol_table, "Pointer to symbol table", 0);
op_hex(pointer_to_symbol_table, 0);
next_ea = pointer_to_symbol_table + 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Number of symbols", 0);
op_hex(next_ea, 0);
next_ea += 4;
uint16_t size_of_optional_header = get_word(next_ea);
create_word(next_ea, 2);
set_cmt(next_ea, "Size of optional header", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Characteristics", 0);
op_hex(next_ea, 0);
next_ea += 2;
add_extra_cmt(next_ea, 1, "IMAGE_OPTIONAL_HEADER");
ea_t image_optional_header = next_ea;
create_word(next_ea, 2);
set_cmt(next_ea, "Magic number", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_byte(next_ea, 1);
```

```
set_cmt(next_ea, "Major linker version", 0);
op_hex(next_ea, 0);
next_ea += 1;
create_byte(next_ea, 1);
set_cmt(next_ea, "Minor linker version", 0);
op_hex(next_ea, 0);
next_ea += 1;
create_dword(next_ea, 4);
set_cmt(next_ea, "Size of code", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Size of initialized data", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Size of uninitialized data", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Address of entry point", 0);
make_entry(get_dword(next_ea));
refinfo_t ri;
ri.init(REF_OFF64, image_base);
if (is_loaded(next_ea) && get_dword(next_ea)) {
    op_offset_ex(next_ea, 0, &ri);
}
next_ea += 4;
create_dword(next_ea, 4);
if (is_loaded(next_ea) && get_dword(next_ea)) {
    op_offset_ex(next_ea, 0, &ri);
}
set_cmt(next_ea, "Base of code", 0);
next_ea += 4;
uint64_t default_image_base = get_qword(next_ea);
create_qword(next_ea, 8);
set_cmt(next_ea, "Image base", 0);
op_hex(next_ea, 0);
op_plain_offset(next_ea, 0, *pe_base);
next_ea += 8;
create_dword(next_ea, 4);
set_cmt(next_ea, "Section alignment", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "File alignment", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_word(next_ea, 2);
set_cmt(next_ea, "Major operating system version", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Minor operating system version", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Major image version", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Minor image version", 0);
op_hex(next_ea, 0);
next_ea += 2;
```

```
create_word(next_ea, 2);
set_cmt(next_ea, "Major subsystem version", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Minor subsystem version", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_dword(next_ea, 4);
set_cmt(next_ea, "Win32 Version value", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Size of image", 0);
op_hex(next_ea, 0);
image_size = get_dword(next_ea);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Size of headers", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Checksum", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_word(next_ea, 2);
set_cmt(next_ea, "Subsystem", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_word(next_ea, 2);
set_cmt(next_ea, "Dll characteristics", 0);
op_hex(next_ea, 0);
next_ea += 2;
create_qword(next_ea, 8);
set_cmt(next_ea, "Size of stack reserve", 0);
op_hex(next_ea, 0);
next_ea += 8;
create_qword(next_ea, 8);
set_cmt(next_ea, "Size of stack commit", 0);
op_hex(next_ea, 0);
next_ea += 8;
create_qword(next_ea, 8);
set_cmt(next_ea, "Size of heap reserve", 0);
op_hex(next_ea, 0);
next_ea += 8;
create_qword(next_ea, 8);
set_cmt(next_ea, "Size of heap commit", 0);
op_hex(next_ea, 0);
next_ea += 8;
create_dword(next_ea, 4);
set_cmt(next_ea, "Loader flag", 0);
op_hex(next_ea, 0);
next_ea += 4;
create_dword(next_ea, 4);
set_cmt(next_ea, "Number of data directories", 0);
op_hex(next_ea, 0);
while (!is_loaded(next_ea)) {
    continue;
}
number_of_dirs = get_dword(next_ea);
uint32_t va = 0;
uint32_t size = 0;
next_ea += 4;
for (int i = 0; i < number_of_dirs; i++) {
```

```
        if (is_reloc_dir(i)) {
            uint32_t relocs_rva = get_dword(next_ea);
            uint32_t relocs_size = get_dword(next_ea + 4);
            if (relocs_rva && relocs_size) {
                ea_t relocs_va = image_base + relocs_rva;
                ea_t relocs_va_end = relocs_va + relocs_size;
                ea_t delta = image_base - default_image_base;
                ea_t block_addr = get_dword(relocs_va);
                ea_t block_size = get_dword(relocs_va + 4);
                while (block_size && relocs_va < relocs_va_end) {
                    ea_t block_base = image_base + block_addr;
                    int block_reloc_count = (block_size - 8) / 2;

                    ea_t block_ptr = relocs_va + 8;
                    while (block_reloc_count--) {
                        uint16_t reloc_value = get_word(block_ptr);
                        uint16_t type = reloc_value & PER_TYPE;
                        uint16_t offset = reloc_value & PER_OFF;
                        if (type == PER_DIR64)
                            add_qword(block_base + offset, delta);
                        else if (type == PER_HIGHLOW)
                            add_dword(block_base + offset, (uint32_t)delta);
                        else if (type == PER_HIGH)
                            add_word(block_base + offset, (uint16_t)(delta >> 16));
                        else if (type == PER_LOW)
                            add_word(block_base + offset, (uint16_t)delta);
                        block_ptr += 2;
                    }
                    relocs_va += block_size;

                    block_addr = get_dword(relocs_va);
                    block_size = get_dword(relocs_va + 4);
                }
            }
        }
        if (is_reloc_dir(i) || is_debug_dir(i)) {
            add_extra_cmt(next_ea, true, "%s", DIRECTORIES[i]);
            create_dword(next_ea, 4);
            create_dword(next_ea + 4, 4);
            op_hex(next_ea, 0);
            op_hex(next_ea + 4, 0);
            set_cmt(next_ea, "Virtual address", 0);
            set_cmt(next_ea + 4, "Size", 0);
        } else {
            create_qword(next_ea, 8);
            set_cmt(next_ea, DIRECTORIES[i], 0);
        }
        next_ea += 8;
    }
    del_items(next_ea, DELIT_EXPAND | DELIT_DELNAMES, 0x28 * number_of_sections);
    for (int i = 0; i < number_of_sections; i++) {
        next_ea = process_section_entry(next_ea);
        memset(seg_name, 0, sizeof(seg_name));
        memset(seg_header_name, 0, sizeof(seg_name));
        memset(image_base_name, 0, sizeof(seg_name));
        memset(section_name, 0, sizeof(seg_name));
    }
}
}
```

efiXloader/pe.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * pe.h
 */

#ifndef EFILOADER_PE_H
#define EFILOADER_PE_H

//
// IDA header
//
#include "ida_core.h"
#include "pe_ida.h"
//
// Utilities
//
#include "utils.h"

#include <typeinf.hpp>

#define PAGE_SIZE 0x1000

#define MZ_SIGN 0x5A4D       // MZ header
#define MAGIC_P32 0x10B      // Normal PE file
#define MAGIC_P32_PLUS 0x20B // 64-bit image
#define PE_SIGN 0x4550       // PE signature

namespace efiloader {

class PE {
  public:
    PE(linput_t *i_li, std::basic_string<char> fname, ea_t *base, ushort *sel_base,
       int ord, uint16_t mt) {
        _image_name = fname.substr(fname.find_last_of("/\\") + 1);
        msg("[efiXloader] image name is %s\n", _image_name.c_str());
        pe_base = base;
        pe_sel_base = sel_base;
        li = i_li;
        utils = new Utils;
        _sec_off = 0;
        _sec_ea = 0;
        _sel = 0;
        _ord = ord;
        inf_set_64bit();
        if (mt == PECPU_ARM64) {
            set_processor_type("arm", SETPROC_LOADER);
        } else {
```

```
            set_processor_type("metapc", SETPROC_LOADER);
        }
        cm_t cm = inf_get_cc_cm() & ~CM_MASK;
        inf_set_cc_cm(cm | CM_N64);
        if (default_compiler() == COMP_UNK) {
            set_compiler_id(COMP_MS);
        }
        reset();
    };
    ~PE() {
        close_linput(li);
        delete utils;
    }
    uint32_t number_of_sections;
    uint32_t number_of_dirs;
    char *name;
    bool is_reloc_dir(uint32_t i) { return i == 5; };
    bool is_debug_dir(uint32_t i) { return i == 6; };
    void set_64_bit_segm_and_rabase(ea_t ea) {
        segment_t *tmp_seg = getseg(ea);
        set_segm_addressing(tmp_seg, 2);
        set_segm_base(tmp_seg, *pe_base);
    }
    void set_64_bit(ea_t ea) {
        segment_t *tmp_seg = getseg(ea);
        set_segm_addressing(tmp_seg, 2);
    };
    bool is_p32();
    bool is_p32_plus();
    bool is_pe();
    bool good();
    bool process();
    uint16_t arch();
    // data processing
    inline size_t make_named_byte(ea_t ea, const char *name, const char *extra = NULL,
                                  size_t count = 1);
    inline size_t make_named_word(ea_t ea, const char *name, const char *extra = NULL,
                                  size_t count = 1);
    inline size_t make_named_dword(ea_t ea, const char *name, const char *extra = NULL,
                                   size_t count = 1);
    inline size_t make_named_qword(ea_t ea, const char *name, const char *extra = NULL,
                                   size_t count = 1);
    inline ea_t skip(ea_t ea, qoff64_t off) { return ea + off; };
    // ida db processing
    void push_to_idb(ea_t start, ea_t end) {
        // Map header
        file2base(li, 0x0, start, start + headers_size, FILEREG_PATCHABLE);
        // Map sections
        for (int i = 0; i < number_of_sections; i++) {
            file2base(li, _sec_headers[i].s_scnptr, start + _sec_headers[i].s_vaddr,
                      start + _sec_headers[i].s_vaddr + _sec_headers[i].s_psize,
                      FILEREG_PATCHABLE);
        }
    };

private:
    qvector<ea_t> segments_ea;
    std::basic_string<char> _full_path;
    std::basic_string<char> _image_name;
    efiloader::Utils *utils;
    linput_t *li;
    qoff64_t head_start();
    qoff64_t head_off;
    qoff64_t _pe_header_off;
```

```
    uint16_t headers_size;
    peheader_t pe;
    peheader64_t pe64;
    uint16_t _sec_num;
    uint16_t _bits;
    qvector<sel_t> selectors;
    qvector<sel_t> data_selectors;
    qvector<qstring> ds_seg_names;
    qvector<qstring> cs_seg_names;
    void reset() { qlseek(li, 0); };
    const char *_machine_name();
    //
    // PE image preprocessing
    //
    void preprocess();
    //
    // sections processing
    //
    qvector<pesection_t> _sec_headers;
    ea_t *pe_base;
    ushort *pe_sel_base;
    ushort _sel;
    ea_t _sec_off;
    ea_t _sec_ea;
    // pe ord
    uval_t _ord;
    ea_t image_base;
    uint32_t image_size;
    qvector<size_t> segm_sizes;
    qvector<size_t> segm_raw_sizes;
    qvector<ea_t> segm_entries;

    int preprocess_sections();
    //
    // functions processing
    //
    void make_entry(ea_t ea);
    void make_code(ea_t ea);
    //
    // segments processing
    //
    qstring code_segm_name;
    qstring data_segm_name;
    sel_t data_segment_sel;
    qvector<segment_t *> segments;
    qvector<qstring> segm_names;
    qvector<qstring> secs_names;
    ea_t process_section_entry(ea_t ea);
    segment_t *make_generic_segment(ea_t seg_ea, ea_t seg_ea_end,
                                    const char *section_name, uint32_t flags);
    segment_t *make_head_segment(ea_t start, ea_t end, const char *name);
    void setup_ds_selector();
};
} // namespace efiloader

enum MachineType { AMD64 = 0x8664, I386 = 0x014C, AARCH64 = 0xaa64 };

#endif // EFILOADER_PE_H
}
```

efiXloader/pe_ida.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * pe_ida.h
 */

#ifndef EFILOADER_PE_IDA_H
#define EFILOADER_PE_IDA_H

/*
 *      Interactive disassembler (IDA).
 *      Version 3.05
 *      Copyright (c) 1990-95 by Ilfak Guilfanov. (2:5020/209@fidonet)
 *      ALL RIGHTS RESERVED.
 *
 */

//
//      Portable Executable file format (MS Windows 95, MS Windows NT)
//

#include "ida_core.h"
#include <stddef.h>
#include <time.h>

#pragma pack(push, 1)
//--------------------------------------------------------------------------
//
// 32-bit Portable EXE Header
//
//--------------------------------------------------------------------------
struct petab_t {
    uint32 rva;  // relative virtual address
    uint32 size; // size
};               // PE va/size array element

template <class pointer_t> struct peheader_tpl {
    int32 signature; // 00 Current value is "PE/0/0".

#define PEEXE_ID 0x4550    // 'PE' followed by two zeroes
#define BPEEXE_ID 0x455042 // Borland's extenson for DPMI'host
#define PLEXE_ID
\
    0x4C50 // 'PL', PharLap TNT DOS-Extender Lite file that uses real mode APIs
#define TEEXE_ID 0x5A56      // 'VZ', EFI Terse Executable
    uint16 machine;          // 04 This field specifies the type of CPU
                             // compatibility required by this image to run.
                             // The values are:
```

```
#define PECPU_UNKNOWN 0x0000 // unknown

#define PECPU_80386 0x014C // 80386
#define PECPU_80486 0x014D // 80486
#define PECPU_80586 0x014E // 80586

#define PECPU_R3000 0x0162     // MIPS Mark I (R2000, R3000)
#define PECPU_R6000 0x0163     // MIPS Mark II (R6000)
#define PECPU_R4000 0x0166     // MIPS Mark III (R4000)
#define PECPU_R10000 0x0168    // MIPS Mark IV (R10000)
#define PECPU_WCEMIPSV2 0x0169 // MIPS little-endian WCE v2
#define PECPU_MIPS16 0x0266    // MIPS16
#define PECPU_MIPSFPU 0x0366   // MIPS with FPU
#define PECPU_MIPSFPU16 0x0466 // MIPS16 with FPU

#define PECPU_ALPHA 0x0184   // DEC Alpha
#define PECPU_ALPHA64 0x0284 // Dec Alpha 64-bit

#define PECPU_SH3 0x01A2    // SH3
#define PECPU_SH3DSP 0x01A3 // SH3DSP
#define PECPU_SH3E 0x01A4   // SH3E
#define PECPU_SH4 0x01A6    // SH4
#define PECPU_SH5 0x01A8    // SH5

#define PECPU_ARM 0x01C0   // ARM
#define PECPU_ARMI 0x01C2  // ARM with Thumb
#define PECPU_ARMV7 0x01C4 // ARMv7 (or higher) Thumb mode only

#define PECPU_AM33 0x01D3 // Matsushita (Panasonic) AM33/MN10300

#define PECPU_PPC 0x01F0   // PowerPC
#define PECPU_PPCFP 0x01F1 // PowerPC with floating-point
#define PECPU_PPCBE 0x01F2 // PowerPC Big-Endian

#define PECPU_M32R 0x9041 // M32R little-endian

#define PECPU_IA64 0x0200    // Intel Itanium IA64
#define PECPU_EPOC 0x0A00    // ARM EPOC
#define PECPU_AMD64 0x8664   // AMD64 (x64)
#define PECPU_ARM64 0xaa64   // ARMv8 in 64-bit mode
#define PECPU_M68K 0x0268    // Motorola 68000 series
#define PECPU_EBC 0x0EBC     // EFI Bytecode
#define PECPU_CEF 0x0CEF     // ?
#define PECPU_CEE 0xC0EE     // ?
#define PECPU_TRICORE 0x0520 // TRICORE (Infineon)

    bool is_64bit_cpu(void) const {
        return machine == PECPU_AMD64 || machine == PECPU_IA64 || machine
== PECPU_ARM64;
    }
    bool is_pc(void) const {
        return machine == PECPU_80386 || machine == PECPU_80486 ||
               machine == PECPU_80586 || machine == PECPU_AMD64;
    }
    bool is_mips(void) const {
        return machine == PECPU_R3000 || machine == PECPU_R6000 ||
               machine == PECPU_R4000 || machine == PECPU_R10000 ||
               machine == PECPU_WCEMIPSV2 || machine == PECPU_MIPS16 ||
               machine == PECPU_MIPSFPU || machine == PECPU_MIPSFPU16;
    }

    bool is_arm(void) const {
        return machine == PECPU_ARM || machine == PECPU_ARMI || machine == PECPU_ARMV7;
    }
```

```
    bool has_code16_bit(void) const { return is_arm() || is_mips(); }

    uint16 nobjs; // 06 This field specifies the number of entries
    // in the Object Table.
    qtime32_t datetime; // 08 Used to store the time and date the file was
    // created or modified by the linker.
    uint32 symtof;  // 0C Symbol Table Offset
    uint32 nsyms;   // 10 Number of Symbols in Symbol Table
    uint16 hdrsize; // 14 This is the number of remaining bytes in the NT
    // header that follow the FLAGS field.
    uint16 flags;        // 16 Flag bits for the image.  0000h Program image.
#define PEF_BRVHI 0x8000 // Big endian: MSB precedes LSB in memory
#define PEF_UP 0x4000    // File should be run only on a UP machine
#define PEF_DLL 0x2000   // Dynamic Link Library (DLL)
#define PEF_SYS 0x1000   // System file
#define PEF_NSWAP
\
    0x0800 // If the image is on network media, fully load it and copy it to the
           // swap file.
#define PEF_SWAP
\
    0x0400              // If image is on removable media,
                        // copy and run from swap file
#define PEF_NODEB 0x0200 // Debugging info stripped
#define PEF_32BIT 0x0100 // 32-bit word machine
#define PEF_BRVLO 0x0080 // Little endian: LSB precedes MSB in memory
#define PEF_16BIT 0x0040 // 16-bit word machine
#define PEF_2GB 0x0020   // App can handle > 2gb addresses
#define PEF_TMWKS 0x0010 // Aggressively trim working set
#define PEF_NOSYM 0x0008 // Local symbols stripped
#define PEF_NOLIN 0x0004 // Line numbers stripped
#define PEF_EXEC 0x0002  // Image is executable
#define PEF_NOFIX 0x0001 // Relocation info stripped

    int32 first_section_pos(int32 peoff) const {
        return peoff + offsetof(peheader_tpl, magic) + hdrsize;
    }

    // COFF fields:

    uint16 magic;            // 18 Magic
#define MAGIC_ROM 0x107      // ROM image
#define MAGIC_P32 0x10B      // Normal PE file
#define MAGIC_P32_PLUS 0x20B // 64-bit image
    bool is_pe_plus(void) const { return magic == MAGIC_P32_PLUS; }
    uchar vstamp_major; // 1A Major Linker Version
    uchar vstamp_minor; // 1B Minor Linker Version
    uint32 tsize;       // 1C TEXT size (padded)
    uint32 dsize;       // 20 DATA size (padded)
    uint32 bsize;       // 24 BSS  size (padded)
    uint32 entry;       // 28 Entry point
    uint32 text_start;  // 2C Base of text
    union {
        struct {
            uint32 data_start; // 30 Base of data

            // Win32/NT extensions:

            uint32 imagebase32; // 34 Virtual base of the image.
        };
        uint64 imagebase64;
    };
    uint64 imagebase() const {
```

```
        if (is_pe_plus())
            return imagebase64;
        else
            return imagebase32;
    }
    // This will be the virtual address of the first
    // byte of the file (Dos Header).  This must be
    // a multiple of 64K.
    uint32 objalign; // 38 The alignment of the objects. This must be a power
    // of 2 between 512 and 256M inclusive. The default
    // is 64K.
    uint32 filealign; // 3C Alignment factor used to align image pages.
                      // The alignment factor (in bytes) used to align the
                      // base of the image pages and to determine the
                      // granularity of per-object trailing zero pad.
                      // Larger alignment factors will cost more file space;
                      // smaller alignment factors will impact demand load
                      // performance, perhaps significantly. Of the two,
                      // wasting file space is preferable.  This value
                      // should be a power of 2 between 512 and 64K inclusive.
                      // Get the file position aligned:
#define FILEALIGN 512 // IDA5.1: it seems that for standard object alignment (if 4096)
    // the Windows kernel does not use filealign
    // (just checks that it is in the valid range) but uses 512
    uint32 get_align_mask(void) const {
        return ((objalign == 4096 || filealign == 0) ? FILEALIGN : filealign) - 1;
    }
    uint32 align_up_in_file(uint32 pos) const {
        if (is_efi()) // apparently EFI images are not aligned
            return pos;
        uint32 mask = get_align_mask();
        return (pos + mask) & ~mask;
    }
    uint32 align_down_in_file(uint32 pos) const {
        return is_efi() ? pos : (pos & ~get_align_mask());
    }
    uint16 osmajor;     // 40 OS version number required to run this image.
    uint16 osminor;     // 42 OS version number required to run this image.
    uint16 imagemajor;  // 44 User major version number.
    uint16 imageminor;  // 46 User minor version number.
    uint16 subsysmajor; // 48 Subsystem major version number.
    uint16 subsysminor; // 4A Subsystem minor version number.

    uint32 subsystem_version(void) const { return (subsysmajor << 16) | subsysminor; }

    uint32 reserved;  // 4C
    uint32 imagesize; // 50 The virtual size (in bytes) of the image.
    // This includes all headers.  The total image size
    // must be a multiple of Object Align.
    uint32 allhdrsize; // 54 Total header size. The combined size of the Dos
    // Header, PE Header and Object Table.
    uint32 checksum;       // 58 Checksum for entire file.  Set to 0 by the linker.
    uint16 subsys;         // 5C NT Subsystem required to run this image.
#define PES_UNKNOWN 0x0000 // Unknown
#define PES_NATIVE 0x0001  // Native
#define PES_WINGUI 0x0002  // Windows GUI
#define PES_WINCHAR 0x0003 // Windows Character
#define PES_OS2CHAR 0x0005 // OS/2 Character
#define PES_POSIX 0x0007   // Posix Character
#define PES_NAT9x 0x0008   // image is a native Win9x driver
#define PES_WINCE 0x0009   // Runs on Windows CE.
#define PES_EFI_APP 0x000A // EFI application.
#define PES_EFI_BDV 0x000B // EFI driver that provides boot services.
#define PES_EFI_RDV 0x000C // EFI driver that provides runtime services.
```

```
#define PES_EFI_ROM 0x000D // EFI ROM image
#define PES_XBOX 0x000E    // Xbox system
#define PES_BOOTAPP 0x0010 // Windows Boot Application

    bool is_efi(void) const {
        return subsys == PES_EFI_APP || subsys == PES_EFI_BDV || subsys ==
PES_EFI_RDV ||
               subsys == PES_EFI_ROM;
    }
    bool is_console_app(void) const {
        return subsys == PES_WINCHAR || subsys == PES_OS2CHAR || subsys == PES_POSIX;
    }
    bool is_userland(void) const {
        return subsys == PES_WINGUI || subsys == PES_WINCHAR || subsys ==
PES_OS2CHAR ||
               subsys == PES_POSIX || subsys == PES_WINCE;
    }
    uint16 dllflags;     // 5E Indicates special loader requirements.
#define PEL_PINIT 0x0001 // Per-Process Library Initialization.
#define PEL_PTERM 0x0002 // Per-Process Library Termination.
#define PEL_TINIT 0x0004 // Per-Thread Library Initialization.
#define PEL_TTERM 0x0008 // Per-Thread Library Termination.
#define PEL_HIGH_ENT
\
    0x0020 // Image can handle a high entropy 64-bit virtual address space.
#define PEL_DYNAMIC_BASE 0x0040    // The DLL can be relocated at load time.
#define PEL_FORCE_INTEGRITY 0x0080 // Code integrity checks are forced.
#define PEF_NX 0x0100 // The image is compatible with data execution prevention (DEP).
#define PEF_NO_ISOLATION
\
    0x0200 // The image is isolation aware, but should not be isolated.
#define PEF_NO_SEH
\
    0x0400 // The image does not use structured exception handling (SEH). No
           // handlers can be called in this image.
#define PEL_NO_BIND 0x0800      // Do not bind image
#define PEL_APPCONTAINER 0x1000 // Image should execute in an AppContainer
#define PEL_WDM_DRV 0x2000      // Driver is a WDM Driver
#define PEL_GUARDCF 0x4000      // Image supports Control Flow Guard checking
#define PEL_TSRVAWA 0x8000      // Image is Terminal Server aware

    pointer_t stackres; // 60 Stack size needed for image. The memory is
    // reserved, but only the STACK COMMIT SIZE is
    // committed. The next page of the stack is a
    // 'guarded page'. When the application hits the
    // guarded page, the guarded page becomes valid,
    // and the next page becomes the guarded page.
    // This continues until the RESERVE SIZE is reached.
    pointer_t stackcom; // 64 Stack commit size.
    pointer_t heapres;  // 68 Size of local heap to reserve.
    pointer_t heapcom;  // 6C Amount to commit in local heap.
    uint32 loaderflags; // 70 ?
    uint32 nrvas;       // 74 Indicates the size of the VA/SIZE array
    // that follows.
    petab_t expdir; // 0 78 Export Directory
    petab_t impdir; // 1 80 Import Directory
    petab_t resdir; // 2 88 Resource Directory
    petab_t excdir; // 3 90 Exception Directory
    petab_t secdir; // 4 98 Security Directory
    //      The Certificate Table entry points to a table of
    //      attribute certificates. These certificates are not
    //      loaded into memory as part of the image. As such,
    //      the first field of this entry, which is normally
    //      an RVA, is a File Pointer instead
```

```
    petab_t reltab; // 5 A0 Relocation Table
    petab_t debdir; // 6 A8 Debug Directory
    petab_t desstr; // 7 B0 Description String
    petab_t cputab; // 8 B8 Machine Value
    petab_t tlsdir; // 9 C0 TLS Directory
    petab_t loddir; // 10 Load Configuration Directory
    petab_t bimtab; // 11 Bound Import Table address and size.
    petab_t iat;    // 12 Import Address Table address and size.
    petab_t didtab; // 13 Address and size of the Delay Import Descriptor.
    petab_t comhdr; // 14 COM+ Runtime Header address and size
    petab_t x00tab; // 15 Reserved

    bool is_te() const { return signature == TEEXE_ID; }
    inline bool has_debdir() const;
};

typedef peheader_tpl<uint32> peheader_t;
typedef peheader_tpl<uint64> peheader64_t;

const size_t total_rvatab_size = sizeof(peheader_t) - offsetof(peheader_t, expdir);
const size_t total_rvatab_count = total_rvatab_size / sizeof(petab_t);

//--------------------------------------------------------------------------
struct diheader_t {
    uint16 signature; // 00 Current value is "DI"
#define DBG_ID 0x4944
    uint16 flags2; // 02 ?? pedump mentions about this
    //    I've never seen something other than 0
    uint16 machine; // 04 This field specifies the type of CPU
    //    compatibility required by this image to run.
    uint16 flags;       // 06 Flag bits for the image.
    qtime32_t datetime; // 08 Used to store the time and date the file was
    //    created or modified by the linker.
    uint32 checksum;  // 12 Checksum
    uint32 imagebase; // 16 Virtual base of the image.
    //    This will be the virtual address of the first
    //    byte of the file (Dos Header).  This must be
    //    a multiple of 64K.
    uint32 imagesize; // 20 The virtual size (in bytes) of the image.
    //    This includes all headers.  The total image size
    //    must be a multiple of Object Align.
    uint32 n_secs;        // 24 Number of sections
    uint32 exp_name_size; // 28 Exported Names Size
    uint32 dbg_dir_size;  // 32 Debug Directory Size
    uint32 reserved[3];   // 36 Reserved fields
};

//--------------------------------------------------------------------------
//
//      S E C T I O N S
//
struct pesection_t {
    char s_name[8];           /* section name */
    uint32 s_vsize;           /* virtual size */
    uint32 s_vaddr;           /* virtual address */
    uint32 s_psize;           /* physical size */
    int32 s_scnptr;           /* file ptr to raw data for section */
    int32 s_relptr;           /* file ptr to relocation */
    int32 s_lnnoptr;          /* file ptr to line numbers */
    uint16 s_nreloc;          /* number of relocation entries */
    uint16 s_nlnno;           /* number of line number entries */
    int32 s_flags;            /* flags */
#define PEST_REG 0x00000000    // obsolete: regular: allocated, relocated, loaded
#define PEST_DUMMY 0x00000001  // obsolete: dummy: not allocated, relocated, not loaded
```

```
#define PEST_NOLOAD 0x00000002 // obsolete: noload: allocated, relocated, not loaded
#define PEST_GROUP 0x00000004  // obsolete: grouped: formed of input sections
#define PEST_PAD 0x00000008    // obsolete: padding: not allocated, not
relocated, loaded
#define PEST_COPY
\
    0x00000010                 // obsolete: copy: for decision function used
                               //    by field update; not
                               //    allocated, not relocated,
                               //    loaded; reloc & lineno
                               //    entries processed normally */
#define PEST_TEXT 0x00000020L   // section contains text only
#define PEST_DATA 0x00000040L   // section contains data only
#define PEST_BSS 0x00000080L    // section contains bss only
#define PEST_EXCEPT 0x00000100L // obsolete: Exception section
#define PEST_INFO 0x00000200L   // Comment: not allocated, not relocated, not loaded
#define PEST_OVER 0x00000400L   // obsolete: Overlay: not allocated, relocated,
not loaded
#define PEST_LIB 0x00000800L    // ".lib" section: treated like PEST_INFO

#define PEST_LOADER 0x00001000L // Loader section: COMDAT
#define PEST_DEBUG 0x00002000L  // Debug section:
#define PEST_TYPCHK 0x00004000L // Type check section:
#define PEST_OVRFLO 0x00008000L // obsolete: RLD and line number overflow sec hdr
#define PEST_F0000 0x000F0000L  // Unknown
#define PEST_ALIGN 0x00F00000L  // Alignment 2^(x-1):
    uint32 get_sect_alignment(void) const {
        int align = ((s_flags >> 20) & 15);
        return align == 0 ? 0 : (1 << (align - 1));
    }

    asize_t get_vsize(const peheader_t &pe) const {
        return align_up(s_vsize ? s_vsize : s_psize, pe.objalign ? pe.objalign : 1);
    }

    asize_t get_psize(const peheader_t &pe) const { return qmin(s_psize,
get_vsize(pe)); }

#define PEST_1000000 0x01000000L // Unknown
#define PEST_DISCARD 0x02000000L // Discardable
#define PEST_NOCACHE 0x04000000L // Not cachable
#define PEST_NOPAGE 0x08000000L  // Not pageable
#define PEST_SHARED 0x10000000L  // Shareable
#define PEST_EXEC 0x20000000L    // Executable
#define PEST_READ 0x40000000L    // Readable
#define PEST_WRITE 0x80000000L   // Writable
};

//-------------------------------------------------------------------------
//
//      E X P O R T S
//
struct peexpdir_t {
    uint32 flags;       // Currently set to zero.
    qtime32_t datetime; // Time/Date the export data was created.
    uint16 major;       // A user settable major/minor version number.
    uint16 minor;
    uint32 dllname; // Relative Virtual Address of the Dll asciiz Name.
    // This is the address relative to the Image Base.
    uint32 ordbase; // First valid exported ordinal. This field specifies
    // the starting ordinal number for the export
    // address table for this image. Normally set to 1.
    uint32 naddrs; // Indicates number of entries in the Export Address
    // Table.
```

```
    uint32 nnames; // This indicates the number of entries in the Name
    // Ptr Table (and parallel Ordinal Table).
    uint32 adrtab; // Relative Virtual Address of the Export Address
    // Table. This address is relative to the Image Base.
    uint32 namtab; // Relative Virtual Address of the Export Name Table
    // Pointers. This address is relative to the
    // beginning of the Image Base. This table is an
    // array of RVA's with # NAMES entries.
    uint32 ordtab; // Relative Virtual Address of Export Ordinals
                   // Table Entry. This address is relative to the
                   // beginning of the Image Base.
};

//-------------------------------------------------------------------------
//
//      I M P O R T S
//
struct peimpdir_t {
    uint32 table1;      // aka OriginalFirstThunk
    qtime32_t datetime; // Time/Date the import data was pre-snapped or
    // zero if not pre-snapped.
    uint32 fchain;  // Forwarder chain
    uint32 dllname; // Relative Virtual Address of the Dll asciiz Name.
    // This is the address relative to the Image Base.
    uint32 looktab; // aka FirstThunk
                    // This field contains the address of the start of
                    // the import lookup table for this image. The address
                    // is relative to the beginning of the Image Base.
#define hibit(type) ((type(-1) >> 1) ^ type(-1))
#define IMP_BY_ORD32 hibit(uint32) // Import by ordinal, otherwise by name
#define IMP_BY_ORD64 hibit(uint64) // Import by ordinal, otherwise by name

    peimpdir_t(void) { memset(this, 0, sizeof(peimpdir_t)); }
};

struct dimpdir_t // delayed load import table
{
    uint32 attrs;          // Attributes.
#define DIMP_NOBASE 0x0001 // pe.imagebase was not added to addresses
    uint32 dllname;        // Relative virtual address of the name of the DLL
    // to be loaded. The name resides in the read-only
    // data section of the image.
    uint32 handle; // Relative virtual address of the module handle
    // (in the data section of the image) of the DLL to
    // be delay-loaded. Used for storage by the routine
    // supplied to manage delay-loading.
    uint32 diat; // Relative virtual address of the delay-load import
    // address table. See below for further details.
    uint32 dint; // Relative virtual address of the delay-load name
    // table, which contains the names of the imports
    // that may need to be loaded. Matches the layout of
    // the Import Name Table, Section 6.4.3. Hint/Name Table.
    uint32 dbiat; // Bound Delay Import Table. Relative virtual address
    // of the bound delay-load address table, if it exists.
    uint32 duiat; // Unload Delay Import Table. Relative virtual address
    // of the unload delay-load address table, if it exists.
    // This is an exact copy of the Delay Import Address
    // Table. In the event that the caller unloads the DLL,
    // this table should be copied back over the Delay IAT
    // such that subsequent calls to the DLL continue to
    // use the thunking mechanism correctly.
    qtime32_t datetime; // Time stamp of DLL to which this image has been bound.
};
```

```
// Bound Import Table format:

struct BOUND_IMPORT_DESCRIPTOR {
    qtime32_t TimeDateStamp;
    uint16 OffsetModuleName;
    uint16 NumberOfModuleForwarderRefs;
    // Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
};

struct BOUND_FORWARDER_REF {
    qtime32_t TimeDateStamp;
    uint16 OffsetModuleName;
    uint16 Reserved;
};

//-------------------------------------------------------------------------
//
//      T H R E A D   L O C A L   D A T A
//

struct image_tls_directory64 {
    uint64 StartAddressOfRawData;
    uint64 EndAddressOfRawData;
    uint64 AddressOfIndex;     // PDWORD
    uint64 AddressOfCallBacks; // PIMAGE_TLS_CALLBACK *;
    uint32 SizeOfZeroFill;
    uint32 Characteristics;
};

struct image_tls_directory32 {
    uint32 StartAddressOfRawData;
    uint32 EndAddressOfRawData;
    uint32 AddressOfIndex;     // PDWORD
    uint32 AddressOfCallBacks; // PIMAGE_TLS_CALLBACK *
    uint32 SizeOfZeroFill;
    uint32 Characteristics;
};

//-------------------------------------------------------------------------
//
//      Exception Tables (.pdata)
//

// ARM, PowerPC, SH3 and SH4 WindowsCE platforms
struct function_entry_ce {
    uint32 FuncStart;         // Virtual address of the corresponding function.
    uint32 PrologLen : 8;     // Number of instructions in the function's prolog.
    uint32 FuncLen : 22;      // Number of instructions in the function.
    uint32 ThirtyTwoBit : 1;  // Set if the function is comprised of 32-bit
                              // instructions, cleared for a 16-bit function.
    uint32 ExceptionFlag : 1; // Set if an exception handler exists for the
                              // function.
};

// ARMv7
struct function_entry_armv7 {
    uint32 BeginAddress; // The RVA of the corresponding function
    uint32 UnwindInfo;   // The RVA of the unwind information, including function
                         // length.
    // If the low 2 bits are non-zero, then this word represents a
    // compacted inline form of the unwind information,
    // including function length.
};
```

```
// for MIPS and 32-bit Alpha
struct function_entry_alpha {
    uint32 BeginAddress;     // Virtual address of the corresponding function.
    uint32 EndAddress;       // Virtual address of the end of the function.
    uint32 ExceptionHandler; // Pointer to the exception handler to be executed.
    uint32 HandlerData;      // Pointer to additional information to be passed to the
                             // handler.
    uint32 PrologEndAddress; // Virtual address of the end of the function's
                             // prolog.
};

// x64
typedef enum _UNWIND_OP_CODES {
    UWOP_PUSH_NONVOL = 0,     // info == register number
    UWOP_ALLOC_LARGE = 1,     // alloc size/8 in next 1(info=0) or 2(info=1) slots
    UWOP_ALLOC_SMALL = 2,     // info == size of allocation / 8 - 1
    UWOP_SET_FPREG = 3,       // FP = RSP + UNWIND_INFO.FPRegOffset*16
    UWOP_SAVE_NONVOL = 4,     // info == register number, offset/8 in next slot
    UWOP_SAVE_NONVOL_FAR = 5, // info == register number, offset/8 in next 2 slots
    UWOP_SAVE_XMM = 6,        // Version 1: info == XMM reg number, offset/8 in
next slot
    UWOP_EPILOG = 6,          // Version 2; code offset is epilog size;
    UWOP_SAVE_XMM_FAR = 7, // version 1:info == XMM reg number, offset/8 in next
2 slots
    UWOP_SPARE_CODE = 7,   // unused ("previously 64-bit UWOP_SAVE_XMM_FAR"); skip
2 slots
    UWOP_SAVE_XMM128 = 8,  // info == XMM reg number, offset/16 in next slot
    UWOP_SAVE_XMM128_FAR = 9, // info == XMM reg number, offset/16 in next 2 slots
    UWOP_PUSH_MACHFRAME = 10, // info == 0: no error-code, 1: with error code
} UNWIND_CODE_OPS;

// Define unwind information flags.
//

#define UNW_FLAG_NHANDLER 0x0
#define UNW_FLAG_EHANDLER 0x1
#define UNW_FLAG_UHANDLER 0x2
#define UNW_FLAG_CHAININFO 0x4

//------------------------------------------------------------------------
//
//      F I X U P S
//
struct pefixup_t {
    uint32 page; // The image base plus the page rva is added to each offset
    // to create the virtual address of where the fixup needs to
    // be applied.
    uint32 size; // Number of bytes in the fixup block. This includes the
                 // PAGE RVA and SIZE fields.
};

#define PER_OFF 0x0FFF
#define PER_TYPE 0xF000
#define PER_ABS 0x0000  // This is a NOP. The fixup is skipped.
#define PER_HIGH 0x1000 // Add the high 16-bits of the delta to the
// 16-bit field at Offset. The 16-bit field
// represents the high value of a 32-bit word.
#define PER_LOW 0x2000 // Add the low 16-bits of the delta to the
// 16-bit field at Offset. The 16-bit field
// represents the low half value of a
// 32-bit word.  This fixup will only be
// emitted for a RISC machine when the image
// Object Align isn't the default of 64K.
#define PER_HIGHLOW 0x3000 // Apply the 32-bit delta to the 32-bit field
```

```
// at Offset.
#define PER_HIGHADJUST 0x4000 // This fixup requires a full 32-bit value.
// The high 16-bits is located at Offset, and
// the low 16-bits is located in the next
// Offset array element (this array element
// is included in the SIZE field). The two
// need to be combined into a signed variable.
// Add the 32-bit delta. Then add 0x8000 and
// store the high 16-bits of the signed
// variable to the 16-bit field at Offset.
#define PER_REL5000 0x5000 // Machine-specific
#define PER_SECTION 0x6000 // Reserved for future use
#define PER_REL32 0x7000   // Relative intrasection
#define PER_REL7000 0x7000 // Machine-specific
#define PER_REL8000 0x8000 // Machine-specific
#define PER_REL9000 0x9000 // Machine-specific
#define PER_DIR64 0xA000   // This fixup applies the delta to the 64-bit
// field at Offset
#define PER_HIGH3ADJ 0xB000 // The fixup adds the high 16 bits of the delta
// to the 16-bit field at Offset. The 16-bit
// field represents the high value of a 48-bit
// word. The low 32 bits of the 48-bit value are
// stored in the 32-bit word that follows this
// base relocation. This means that this base
// relocation occupies three slots.

// Platform-specific based relocation types.

#define PER_IA64_IMM64 0x9000

#define PER_MIPS_JMPADDR 0x5000   // base relocation applies to a MIPS
jump instruction.
#define PER_MIPS_JMPADDR16 0x9000 // base relocation applies to a MIPS16
jump instruction.

#define PER_ARM_MOV32A 0x5000 // base relocation applies the difference to the
// 32-bit value encoded in the immediate fields of
// a contiguous MOVW+MOVT pair in ARM mode at offset.
#define PER_ARM_MOV32T 0x7000 // base relocation applies the difference to the
// 32-bit value encoded in the immediate fields of
// a contiguous MOVW+MOVT pair in Thumb mode at offset.

//-------------------------------------------------------------------------
//
//      DBG file debug entry format
//
struct debug_entry_t {
    uint32 flags; // usually zero
    qtime32_t datetime;
    uint16 major;
    uint16 minor;
    int32 type;
#define DBG_COFF 1
#define DBG_CV 2
#define DBG_FPO 3
#define DBG_MISC 4
#define DBG_EXCEPTION 5
#define DBG_FIXUP 6
#define DBG_OMAP_TO_SRC 7
#define DBG_OMAP_FROM_SRC 8
#define DBG_BORLAND 9
#define DBG_RES10 10
#define DBG_CLSID 11
#define DBG_VCFEATURE 12
```

```
#define DBG_POGO 13
#define DBG_ILTCG 14
#define DBG_MPX 15
    uint32 size;
    uint32 rva;  // virtual address
    uint32 seek; // ptr to data in the file
};

// now we can define has_debdir() because we have debug_entry_t defined
template <> inline bool peheader_t::has_debdir() const {
    return debdir.size >= sizeof(debug_entry_t) && debdir.rva != 0;
}

//--------------------------------------------------------------------------
//
//      DBG file COFF debug information header
//
struct coff_debug_t {
    uint32 NumberOfSymbols;
    uint32 LvaToFirstSymbol;
    uint32 NumberOfLinenumbers;
    uint32 LvaToFirstLinenumber;
    uint32 RvaToFirstByteOfCode;
    uint32 RvaToLastByteOfCode;
    uint32 RvaToFirstByteOfData;
    uint32 RvaToLastByteOfData;
};

//--------------------------------------------------------------------------
//
//      DBG file FPO debug information
//
struct fpo_t {
    uint32 address;
    uint32 size;
    uint32 locals;
    uint16 params;
    uchar prolog;
    uchar regs;
#define FPO_REGS 0x07 // register number
#define FPO_SEH 0x08  //
#define FPO_BP 0x10   // has BP frame?
#define FPO_TYPE 0xC0
#define FPO_T_FPO 0x00
#define FPO_T_TRAP 0x40
#define FPO_T_TSS 0x80
#define FPO_T_NONFPO 0xC0
};

//      DBG file OMAP debug information

struct omap_t {
    uint32 a1;
    uint32 a2;
};

// misc entry format
struct misc_debug_t {
    uint32 type; // type of misc data, see defines
#define MISC_EXENAME 1
    uint32 length; // total length of record, rounded to four
    // byte multiple.
    uchar unicode;     // TRUE if data is unicode string
    uchar reserved[3]; // padding
```

```
    uchar data[1];     // Actual data
};

//--------------------------------------------------------------------
// Resource information
struct rsc_dir_t {
    uint32 Characteristics;
    uint32 TimeDateStamp;
    uint16 MajorVersion;
    uint16 MinorVersion;
    uint16 NumberOfNamedEntries;
    uint16 NumberOfIdEntries;
};

struct rsc_dir_entry_t {
    union {
        struct {
            uint32 NameOffset : 31;
            uint32 NameIsString : 1;
        };
        uint32 Name;
        uint16 Id;
    };
    union {
        uint32 OffsetToData;
        struct {
            uint32 OffsetToDirectory : 31;
            uint32 DataIsDirectory : 1;
        };
    };
};

struct rsc_data_entry_t {
    uint32 OffsetToData;
    uint32 Size;
    uint32 CodePage;
    uint32 Reserved;
};

// Resource types
#define PE_RT_CURSOR 1
#define PE_RT_BITMAP 2
#define PE_RT_ICON 3
#define PE_RT_MENU 4
#define PE_RT_DIALOG 5
#define PE_RT_STRING 6
#define PE_RT_FONTDIR 7
#define PE_RT_FONT 8
#define PE_RT_ACCELERATOR 9
#define PE_RT_RCDATA 10
#define PE_RT_MESSAGETABLE 11
#define PE_RT_GROUP_CURSOR 12
#define PE_RT_GROUP_ICON 14
#define PE_RT_VERSION 16
#define PE_RT_DLGINCLUDE 17
#define PE_RT_PLUGPLAY 19
#define PE_RT_VXD 20
#define PE_RT_ANICURSOR 21
#define PE_RT_ANIICON 22
#define PE_RT_HTML 23
#define PE_RT_MANIFEST 24

// Language codes
#define PE_LANG_NEUTRAL 0x00
```

```
#define PE_LANG_INVARIANT 0x7f

#define PE_LANG_AFRIKAANS 0x36
#define PE_LANG_ALBANIAN 0x1c
#define PE_LANG_ARABIC 0x01
#define PE_LANG_ARMENIAN 0x2b
#define PE_LANG_ASSAMESE 0x4d
#define PE_LANG_AZERI 0x2c
#define PE_LANG_BASQUE 0x2d
#define PE_LANG_BELARUSIAN 0x23
#define PE_LANG_BENGALI 0x45
#define PE_LANG_BULGARIAN 0x02
#define PE_LANG_CATALAN 0x03
#define PE_LANG_CHINESE 0x04
#define PE_LANG_CROATIAN 0x1a
#define PE_LANG_CZECH 0x05
#define PE_LANG_DANISH 0x06
#define PE_LANG_DIVEHI 0x65
#define PE_LANG_DUTCH 0x13
#define PE_LANG_ENGLISH 0x09
#define PE_LANG_ESTONIAN 0x25
#define PE_LANG_FAEROESE 0x38
#define PE_LANG_FARSI 0x29
#define PE_LANG_FINNISH 0x0b
#define PE_LANG_FRENCH 0x0c
#define PE_LANG_GALICIAN 0x56
#define PE_LANG_GEORGIAN 0x37
#define PE_LANG_GERMAN 0x07
#define PE_LANG_GREEK 0x08
#define PE_LANG_GUJARATI 0x47
#define PE_LANG_HEBREW 0x0d
#define PE_LANG_HINDI 0x39
#define PE_LANG_HUNGARIAN 0x0e
#define PE_LANG_ICELANDIC 0x0f
#define PE_LANG_INDONESIAN 0x21
#define PE_LANG_ITALIAN 0x10
#define PE_LANG_JAPANESE 0x11
#define PE_LANG_KANNADA 0x4b
#define PE_LANG_KASHMIRI 0x60
#define PE_LANG_KAZAK 0x3f
#define PE_LANG_KONKANI 0x57
#define PE_LANG_KOREAN 0x12
#define PE_LANG_KYRGYZ 0x40
#define PE_LANG_LATVIAN 0x26
#define PE_LANG_LITHUANIAN 0x27
#define PE_LANG_MACEDONIAN 0x2f // the Former Yugoslav Republic of Macedonia
#define PE_LANG_MALAY 0x3e
#define PE_LANG_MALAYALAM 0x4c
#define PE_LANG_MANIPURI 0x58
#define PE_LANG_MARATHI 0x4e
#define PE_LANG_MONGOLIAN 0x50
#define PE_LANG_NEPALI 0x61
#define PE_LANG_NORWEGIAN 0x14
#define PE_LANG_ORIYA 0x48
#define PE_LANG_POLISH 0x15
#define PE_LANG_PORTUGUESE 0x16
#define PE_LANG_PUNJABI 0x46
#define PE_LANG_ROMANIAN 0x18
#define PE_LANG_RUSSIAN 0x19
#define PE_LANG_SANSKRIT 0x4f
#define PE_LANG_SINDHI 0x59
#define PE_LANG_SLOVAK 0x1b
#define PE_LANG_SLOVENIAN 0x24
#define PE_LANG_SPANISH 0x0a
```

```
#define PE_LANG_SWAHILI 0x41
#define PE_LANG_SWEDISH 0x1d
#define PE_LANG_SYRIAC 0x5a
#define PE_LANG_TAMIL 0x49
#define PE_LANG_TATAR 0x44
#define PE_LANG_TELUGU 0x4a
#define PE_LANG_THAI 0x1e
#define PE_LANG_TURKISH 0x1f
#define PE_LANG_UKRAINIAN 0x22
#define PE_LANG_URDU 0x20
#define PE_LANG_UZBEK 0x43
#define PE_LANG_VIETNAMESE 0x2a

//------------------------------------------------------------------------

#define PE_NODE "$ PE header" // netnode name for PE header
// value()        -> peheader_t
// altval(segnum) -> s->start_ea
#define PE_ALT_DBG_FPOS nodeidx_t(-1) // altval() -> translated fpos of debuginfo
#define PE_ALT_IMAGEBASE
\
    nodeidx_t(-2) // altval() -> loading address (usually pe.imagebase)
#define PE_ALT_PEHDR_OFF nodeidx_t(-3) // altval() -> offset of PE header
#define PE_ALT_NEFLAGS nodeidx_t(-4)   // altval() -> neflags
#define PE_ALT_TDS_LOADED
\
    nodeidx_t(-5) // altval() -> tds already loaded(1) or invalid(-1)
#define PE_ALT_PSXDLL
\
    nodeidx_t(-6) // altval() -> if POSIX(x86) imports from PSXDLL netnode
#define PE_ALT_OVRVA nodeidx_t(-7)    // altval() -> overlay rva (if present)
#define PE_ALT_OVRSZ nodeidx_t(-8)    // altval() -> overlay size (if present)
#define PE_SUPSTR_PDBNM nodeidx_t(-9) // supstr() -> pdb file name
// supval(segnum) -> pesection_t
// blob(0, PE_NODE_RELOC)  -> relocation info
// blob(0, RSDS_TAG)  -> rsds_t structure
// blob(0, NB10_TAG)  -> cv_info_pdb20_t structure
#define PE_ALT_NTAPI nodeidx_t(-10) // altval() -> uses Native API
#define PE_NODE_RELOC 'r'
#define RSDS_TAG 's'
#define NB10_TAG 'n'
#define UTDS_TAG 't'

struct load_config_t {
    uint32 Size;
    uint32 TimeDateStamp;
    uint16 MajorVersion;
    uint16 MinorVersion;
    uint32 GlobalFlagsClear;
    uint32 GlobalFlagsSet;
    uint32 CriticalSectionDefaultTimeout;
    uint32 DeCommitFreeBlockThreshold;
    uint32 DeCommitTotalFreeThreshold;
    uint32 LockPrefixTable; // VA
    uint32 MaximumAllocationSize;
    uint32 VirtualMemoryThreshold;
    uint32 ProcessHeapFlags;
    uint32 ProcessAffinityMask;
    uint16 CSDVersion;
    uint16 Reserved1;
    uint32 EditList;       // VA
    uint32 SecurityCookie; // VA
    // Version 2
    uint32 SEHandlerTable; // VA
```

```
    uint32 SEHandlerCount;
    // Version 3
    uint32 GuardCFCheckFunctionPointer;    // VA
    uint32 GuardCFDispatchFunctionPointer; // VA
    uint32 GuardCFFunctionTable;           // VA
    uint32 GuardCFFunctionCount;
    uint32 GuardFlags;
};

struct load_config64_t {
    uint32 Size;
    uint32 TimeDateStamp;
    uint16 MajorVersion;
    uint16 MinorVersion;
    uint32 GlobalFlagsClear;
    uint32 GlobalFlagsSet;
    uint32 CriticalSectionDefaultTimeout;
    uint64 DeCommitFreeBlockThreshold;
    uint64 DeCommitTotalFreeThreshold;
    uint64 LockPrefixTable; // VA
    uint64 MaximumAllocationSize;
    uint64 VirtualMemoryThreshold;
    uint64 ProcessAffinityMask;
    uint32 ProcessHeapFlags;
    uint16 CSDVersion;
    uint16 Reserved1;
    uint64 EditList;       // VA
    uint64 SecurityCookie; // VA
    // Version 2
    uint64 SEHandlerTable; // VA
    uint64 SEHandlerCount;
    // Version 3
    uint64 GuardCFCheckFunctionPointer;    // VA
    uint64 GuardCFDispatchFunctionPointer; // VA
    uint64 GuardCFFunctionTable;           // VA
    uint64 GuardCFFunctionCount;
    uint32 GuardFlags;
};

#ifndef IMAGE_GUARD_CF_INSTRUMENTED
#define IMAGE_GUARD_CF_INSTRUMENTED
\
    0x000000100 // Module performs control flow integrity checks using
                // system-supplied support
#define IMAGE_GUARD_CFW_INSTRUMENTED
\
    0x000000200 // Module performs control flow and write integrity checks
#define IMAGE_GUARD_CF_FUNCTION_TABLE_PRESENT
\
    0x000000400 // Module contains valid control flow target metadata
#define IMAGE_GUARD_SECURITY_COOKIE_UNUSED
\
    0x000000800 // Module does not make use of the /GS security cookie
#define IMAGE_GUARD_PROTECT_DELAYLOAD_IAT
\
    0x00001000 // Module supports read only delay load IAT
#define IMAGE_GUARD_DELAYLOAD_IAT_IN_ITS_OWN_SECTION
\
    0x00002000 // Delayload import table in its own .didat section (with nothing
               // else in it) that can be freely reprotected
#define IMAGE_GUARD_CF_FUNCTION_TABLE_SIZE_MASK
\
    0xF0000000 // Stride of Guard CF function table encoded in these bits
               // (additional count of bytes per element)
```

```
#define IMAGE_GUARD_CF_FUNCTION_TABLE_SIZE_SHIFT
\
    28 // Shift to right-justify Guard CF function table stride
#endif

//------------------------------------------------------------------------
// MS Windows CLSID, GUID
struct clsid_t {
    uint32 id1;
    uint16 id2;
    uint16 id3;
    uchar id4[8];
    bool operator==(const struct clsid_t &r) const {
        return memcmp(this, &r, sizeof(r)) == 0;
    }
};

//------------------------------------------------------------------------
// RSDS debug information
struct rsds_t {
    uint32 magic;
#define RSDS_MAGIC MC4('R', 'S', 'D', 'S')
#define UTDS_MAGIC MC4('u', 'T', 'D', 'S')
    clsid_t guid;
    uint32 age;
    //  char name[];  // followed by a zero-terminated UTF8 file name
};

//------------------------------------------------------------------------
// NB10 debug information
struct cv_info_pdb20_t {
    uint32 magic; // 'NB10'
#define NB10_MAGIC MC4('N', 'B', '1', '0')
    uint32 offset;
    uint32 signature;
    uint32 age;
    // char pdb_file_name[];
};

//------------------------------------------------------------------------
// MTOC debug information.
// denotes EFI binaries that were built on OSX as Mach-O, then converted to PE
// by the 'mtoc' utility. see
// https://opensource.apple.com/source/cctools/cctools-921/efitools/mtoc.c.auto.html
struct mtoc_info_t {
    uint32 magic; // 'MTOC'
#define MTOC_MAGIC MC4('M', 'T', 'O', 'C')
    uchar uuid[16]; // UUID of original Mach-O file
                    // char debug_filename[];
};

// TE (Terse Executable)
struct teheader_t {
    uint16 signature; // 00
    uint16 machine;   // 02 same as in PE

    bool is_64bit_cpu(void) const {
        return machine == PECPU_AMD64 || machine == PECPU_IA64 || machine
== PECPU_ARM64;
    }

    uint8 nobjs;         // 04 number of sections
    uint8 subsys;        // 05 target subsystem
    uint16 strippedsize; // 06 number of bytes removed from the base of the
```

```
                        // original image

    int32 first_section_pos(int32 peoff) const { return peoff + sizeof(teheader_t); }

    // value which should be added to the sections' file offsets and RVAs
    int32 te_adjust() const { return sizeof(teheader_t) - strippedsize; }

    uint32 entry;       // 08 Entry point
    uint32 text_start;  // 0C Base of code
    uint64 imagebase64; // 10 Virtual base of the image.
    uint64 imagebase() const { return imagebase64; }
    petab_t reltab; // 18 Relocation Table
    petab_t debdir; // 20 Debug Directory
};

const char *get_pe_machine_name(uint16 machine);
void print_pe_flags(uint16 flags);

#pragma pack(pop)

#endif // EFILOADER_PE_IDA_H
}
```

efiXloader/pe_manager.cpp

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * pe_manager.cpp
 */

#include "pe_manager.h"

void efiloader::PeManager::process(linput_t *li, std::basic_string<char> fname, int
ord) {
    efiloader::PE pe(li, fname, &pe_base, &pe_sel_base, ord, machine_type);
    if (pe.good() && pe.is_p32_plus()) {
        msg("[efiXloader] PE detected\n");
        pe.process();
    } else if (pe.is_p32()) {
        msg("[efiXloader] this loader is not ready for PE32\n");
    } else {
        warning("[efiXloader] not PE\n");
    }
```

```
}
}
```

efiXloader/pe_manager.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * pe_manager.h
 */

#ifndef EFILOADER_PE_MANAGER_H
#define EFILOADER_PE_MANAGER_H

#include "ida_core.h"
#include "pe.h"

namespace efiloader {
class PeManager {
  public:
    PeManager(uint16_t mt) {
        inf_set_64bit();
        set_imagebase(0x0);
        if (mt == PECPU_ARM64) {
            set_processor_type("arm", SETPROC_LOADER);
        } else {
            set_processor_type("metapc", SETPROC_LOADER);
        }
        pe_base = 0;
        pe_sel_base = 0;
        machine_type = mt;
    };
    void process(linput_t *li, std::basic_string<char> fname, int ord);
    uint16_t machine_type;

  private:
    void to_base(linput_t *);
    efiloader::PE *pe;
    qvector<efiloader::PE *> pe_files;
    ushort pe_sel_base;
    ea_t pe_base;
    // head processing
    void pe_head_to_base(linput_t *li);
};
} // namespace efiloader
```

```
#endif // EFILOADER_PE_MANAGER_H
}
```

efiXloader/uefitool.cpp

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * uefitool.cpp
 */

#include "uefitool.h"
#include <codecvt>
#include <filesystem>
#include <vector>

void efiloader::File::print() {
    msg("[UEFITOOL PARSER] file ( %s )  \n", qname.c_str());
    for (int i = 0; i < 0x10; i++) {
        msg("%02X ", ubytes[i]);
    };
    msg("\n");
}

void efiloader::Uefitool::show_messages() {
    for (size_t i = 0; i < messages.size(); i++) {
        msg("[UEFITOOL PARSER] %s\n", messages[i].first.toLocal8Bit());
    }
}

void efiloader::Uefitool::get_unique_name(qstring &name) {
    // If the given name is already in use, create a new one
    qstring new_name = name;
    std::string suf;
    int index = 0;
    while (!(unique_names.insert(new_name).second)) {
        suf = "_" + std::to_string(++index);
        new_name = name + static_cast<qstring>(suf.c_str());
    }
    name = new_name;
}

void efiloader::Uefitool::get_image_guid(qstring &image_guid, UModelIndex index) {
    UString guid;
    UModelIndex guid_index;
    switch (model.subtype(model.parent(index))) {
```

```
    case EFI_SECTION_COMPRESSION:
    case EFI_SECTION_GUID_DEFINED:
        guid_index = model.parent(model.parent(index));
        break;
    default:
        guid_index = model.parent(index);
    }
    // get parent header and read GUID
    guid = guidToUString(
        readUnaligned((const EFI_GUID *)(model.header(guid_index).constData())));
    image_guid = reinterpret_cast<char *>(guid.data);
}

std::vector<std::string>
efiloader::Uefitool::parseDepexSectionBody(const UModelIndex &index, UString &parsed) {
    // Adopted from FfsParser::parseDepexSectionBody
    std::vector<std::string> res;

    if (!index.isValid())
        return res;

    UByteArray body = model.body(index);

    // Check data to be present
    if (body.size() < 2) { // 2 is a minimal sane value, i.e TRUE + END
        return res;
    }

    const EFI_GUID *guid;
    const UINT8 *current = (const UINT8 *)body.constData();

    // Special cases of first opcode
    switch (*current) {
    case EFI_DEP_BEFORE:
        if (body.size() != 2 * EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID)) {
            return res;
        }
        guid = (const EFI_GUID *)(current + EFI_DEP_OPCODE_SIZE);
        parsed += UString("\nBEFORE ") + guidToUString(readUnaligned(guid));
        current += EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID);
        if (*current != EFI_DEP_END) {
            return res;
        }
        return res;
    case EFI_DEP_AFTER:
        if (body.size() != 2 * EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID)) {
            return res;
        }
        guid = (const EFI_GUID *)(current + EFI_DEP_OPCODE_SIZE);
        parsed += UString("\nAFTER ") + guidToUString(readUnaligned(guid));
        current += EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID);
        if (*current != EFI_DEP_END) {
            return res;
        }
        return res;
    case EFI_DEP_SOR:
        if (body.size() <= 2 * EFI_DEP_OPCODE_SIZE) {
            return res;
        }
        parsed += UString("\nSOR");
        current += EFI_DEP_OPCODE_SIZE;
        break;
    }
```

```
    // Parse the rest of depex
    while (current - (const UINT8 *)body.constData() < body.size()) {
        switch (*current) {
        case EFI_DEP_BEFORE: {
            return res;
        }
        case EFI_DEP_AFTER: {
            return res;
        }
        case EFI_DEP_SOR: {
            return res;
        }
        case EFI_DEP_PUSH:
            // Check that the rest of depex has correct size
            if ((UINT32)body.size() -
                    (UINT32)(current - (const UINT8 *)body.constData()) <=
                EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID)) {
                parsed.clear();
                return res;
            }
            guid = (const EFI_GUID *)(current + EFI_DEP_OPCODE_SIZE);
            parsed += UString("\nPUSH ") + guidToUString(readUnaligned(guid));
            // Add protocol GUID to result vector
            res.push_back(
                reinterpret_cast<char *>(guidToUString(readUnaligned(guid)).data));
            current += EFI_DEP_OPCODE_SIZE + sizeof(EFI_GUID);
            break;
        case EFI_DEP_AND:
            parsed += UString("\nAND");
            current += EFI_DEP_OPCODE_SIZE;
            break;
        case EFI_DEP_OR:
            parsed += UString("\nOR");
            current += EFI_DEP_OPCODE_SIZE;
            break;
        case EFI_DEP_NOT:
            parsed += UString("\nNOT");
            current += EFI_DEP_OPCODE_SIZE;
            break;
        case EFI_DEP_TRUE:
            parsed += UString("\nTRUE");
            current += EFI_DEP_OPCODE_SIZE;
            break;
        case EFI_DEP_FALSE:
            parsed += UString("\nFALSE");
            current += EFI_DEP_OPCODE_SIZE;
            break;
        case EFI_DEP_END:
            parsed += UString("\nEND");
            current += EFI_DEP_OPCODE_SIZE;
            // Check that END is the last opcode
            if (current - (const UINT8 *)body.constData() < body.size()) {
                parsed.clear();
            }
            break;
        default:
            return res;
            break;
        }
    }

    return res;
}
```

```
std::vector<std::string>
efiloader::Uefitool::parseAprioriRawSection(const UModelIndex &index) {
    // Adopted from FfsParser::parseDepexSectionBody
    std::vector<std::string> res;

    if (!index.isValid())
        return res;

    UByteArray body = model.body(index);

    // Sanity check
    if (body.size() % sizeof(EFI_GUID)) {
        return res;
    }

    UINT32 count = (UINT32)(body.size() / sizeof(EFI_GUID));
    if (count > 0) {
        for (UINT32 i = 0; i < count; i++) {
            const EFI_GUID *guid = (const EFI_GUID *)body.constData() + i;
            res.push_back(
                reinterpret_cast<char *>(guidToUString(readUnaligned(guid)).data));
        }
    }

    return res;
}

void efiloader::Uefitool::set_machine_type(UByteArray pe_body) {
    const char *data = pe_body.constData();
    if (pe_body.size() < 64) {
        return;
    }
    uint32_t _pe_header_off = *(uint32_t *)(data + 0x3c);
    if (pe_body.size() < _pe_header_off + 6) {
        return;
    }
    if (*(uint32_t *)(data + _pe_header_off) == 0x4550) {
        machine_type = *(uint16_t *)(data + _pe_header_off + 4);
        machine_type_detected = true;
    }
}

void efiloader::Uefitool::handle_raw_section(const UModelIndex &index) {
    UModelIndex parent_file = model.findParentOfType(index, Types::File);
    if (!parent_file.isValid()) {
        return;
    }
    UByteArray parent_file_guid(model.header(parent_file).constData(),
sizeof(EFI_GUID));
    if (parent_file_guid == EFI_PEI_APRIORI_FILE_GUID) {
        msg("[efiXloader] PEI Apriori file found\n");
        get_apriori(index, "PEI_APRIORI_FILE");
    }
    if (parent_file_guid == EFI_DXE_APRIORI_FILE_GUID) {
        msg("[efiXloader] DXE Apriori file found\n");
        get_apriori(index, "DXE_APRIORI_FILE");
    }
}

void efiloader::Uefitool::dump(const UModelIndex &index, uint8_t el_type,
                               efiloader::File *file) {
    qstring module_name("");
    qstring guid("");
```

```
        switch (model.subtype(index)) {
        case EFI_SECTION_RAW:
            handle_raw_section(index);
            break;
        case EFI_SECTION_TE:
            file->is_te = true;
            file->ubytes = model.body(index);
            break;
        case EFI_SECTION_PE32:
            file->is_pe = true;
            file->ubytes = model.body(index);
            if (!machine_type_detected) {
                set_machine_type(model.body(index));
            }
            break;
        case EFI_SECTION_USER_INTERFACE:
            file->has_ui = true;
            if (file->is_pe || file->is_te) {
                file->uname = model.body(index);
                utf16_utf8(&module_name,
                           reinterpret_cast<const wchar16_t *>(file->uname.data()));
                if (module_name.size()) {
                    // save image to the images_guids
                    get_image_guid(guid, index);
                    if (images_guids[guid.c_str()]
                            .is_null()) { // check if GUID already exists
                        get_unique_name(module_name);
                        images_guids[guid.c_str()] = module_name.c_str();
                        file->qname.swap(module_name);
                        file->write();
                        files.push_back(file);
                    }
                }
            }
            break;
        case EFI_SECTION_COMPRESSION:
        case EFI_SECTION_GUID_DEFINED:
            for (int i = 0; i < model.rowCount(index); i++) {
                dump(index.child(i, 0), i, file);
            }
            break;
        // Get DEPEX information
        case EFI_SECTION_DXE_DEPEX:
            get_deps(index, "EFI_SECTION_DXE_DEPEX");
            break;
        case EFI_SECTION_MM_DEPEX:
            get_deps(index, "EFI_SECTION_MM_DEPEX");
            break;
        case EFI_SECTION_PEI_DEPEX:
            get_deps(index, "EFI_SECTION_PEI_DEPEX");
            break;
        case EFI_SECTION_VERSION:
            break;
        default:
            // if there is no UI section, then the image name is GUID
            if ((file->is_pe || file->is_te) && !file->has_ui) {
                get_image_guid(module_name, index);
                file->qname.swap(module_name);
                file->write();
                files.push_back(file);
                if (module_name.size()) {
                    // save image to the images_guids
                    images_guids[module_name.c_str()] = module_name.c_str();
                }
```

```
        }
        break;
    }

    return dump(index);
}

void efiloader::Uefitool::dump(const UModelIndex &index) {
    USTATUS err;
    msg("[UEFITOOL PARSER] file (%s, %s)\n", itemTypeToUString(model.type(index)).data,
        itemSubtypeToUString(model.type(index), model.subtype(index)).data);
    msg("[UEFITOOL PARSER] number of items: %#x\n", model.rowCount(index));
    if (is_file_index(index)) {
        efiloader::File *file = new File;
        for (int i = 0; i < model.rowCount(index); i++) {
            dump(index.child(i, 0), i, file);
        }
    } else {
        for (int i = 0; i < model.rowCount(index); i++) {
            dump(index.child(i, 0));
        }
    }
}

void efiloader::Uefitool::dump() { return dump(model.index(0, 0)); }

void efiloader::Uefitool::get_deps(UModelIndex index, std::string key) {
    UString parsed;
    std::vector<std::string> deps;
    qstring image_guid("");

    get_image_guid(image_guid, index);
    deps = parseDepexSectionBody(index, parsed);
    if (deps.size()) {
        msg("[efiXloader] dependency section for image with GUID %s: %s\n",
            image_guid.c_str(), parsed.data);
        all_deps[key][image_guid.c_str()] = deps;
    }
}

void efiloader::Uefitool::get_apriori(UModelIndex index, std::string key) {
    if (all_deps.contains(key)) {
        return;
    }
    std::vector<std::string> deps = parseAprioriRawSection(index);
    if (deps.empty()) {
        return;
    }
    all_deps[key] = deps;
}

void efiloader::Uefitool::dump_jsons() {
    // Dump deps
    std::filesystem::path out;
    out /= get_path(PATH_TYPE_IDB);
    out.replace_extension(".deps.json");
    std::ofstream out_deps(out);
    out_deps << std::setw(4) << all_deps << std::endl;
    // Dump images
    out.replace_extension("").replace_extension(".images.json");
    std::ofstream out_guids(out);
    out_guids << std::setw(4) << images_guids << std::endl;
}
}
```

efiXloader/uefitool.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * uefitool.h
 */

#ifndef EFILOADER_UEFITOOL_H
#define EFILOADER_UEFITOOL_H

#include "3rd/uefitool/common/LZMA/LzmaCompress.h"
#include "3rd/uefitool/common/LZMA/LzmaDecompress.h"
#include "3rd/uefitool/common/Tiano/EfiTianoCompress.h"
#include "3rd/uefitool/common/Tiano/EfiTianoDecompress.h"
#include "3rd/uefitool/common/basetypes.h"
#include "3rd/uefitool/common/ffs.h"
#include "3rd/uefitool/common/ffsparser.h"
#include "3rd/uefitool/common/ffsreport.h"
#include "3rd/uefitool/common/filesystem.h"
#include "3rd/uefitool/common/guiddatabase.h"
#include "3rd/uefitool/common/treeitem.h"
#include "3rd/uefitool/common/treemodel.h"
#include "3rd/uefitool/common/ustring.h"
#include "3rd/uefitool/version.h"

#include "3rd/uefitool/UEFIExtract/ffsdumper.h"
#include "3rd/uefitool/UEFIExtract/uefidump.h"
#include "fstream"
#include "json.hpp"

#include "ida_core.h"

#include <set>
#ifdef _WIN32
#include <direct.h>
#else
#include <sys/stat.h>
#endif

using namespace nlohmann;

enum FILE_SECTION_TYPE {
    PE_DEPENDENCY_SECTION = 0,
    PE_TE_IMAGE_SECTION = 1,
```

```
    UI_SECTION = 2,
    VERSION_SECTION = 3
};

namespace efiloader {
class File {
  public:
    File() {}
    void set_data(char *data_in, uint32_t size_in) {
        qname.qclear();
        bytes.resize(size_in);
        memcpy(&bytes[0], data_in, size_in);
    };
    void write() {
        qstring idb_path(get_path(PATH_TYPE_IDB));
        qstring images_path = idb_path + qstring(".efiloader");
#ifdef WIN32
        _mkdir(images_path.c_str());
#else
        mkdir(images_path.c_str(), S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
#endif
        if (!qname.empty()) {
            qstring image_path = images_path + qstring("/") + qstring(qname.c_str());
            std::ofstream file;
            file.open(image_path.c_str(), std::ios::out | std::ios::binary);
            file.write(ubytes.constData(), ubytes.size());
            file.close();
            dump_name.swap(image_path);
        }
    }
    void print();
    UByteArray ubytes;
    UByteArray uname;
    bytevec_t bytes;
    char *data = NULL;
    uint32_t size = 0;
    std::string name_utf8;
    std::string name_utf16;
    qstring qname;
    qstring dump_name;
    bool is_pe = false;
    bool is_te = false;
    bool has_ui = false;
};

class Uefitool {
  public:
    Uefitool(bytevec_t &data) {
        buffer = (const char *)&data[0];
        buffer_size = data.size();
        UByteArray ubuffer(buffer, buffer_size);
        FfsParser ffs(&model);
        if (ffs.parse(ubuffer)) {
            loader_failure("failed to parse data via UEFITool");
        }
        messages = ffs.getMessages();
    }
    ~Uefitool(){};
    void show_messages();
    bool messages_occurs() { return !messages.empty(); };
    void dump();
    void dump(const UModelIndex &index);
    void dump(const UModelIndex &index, uint8_t el_type, File *pe_file);
    void handle_raw_section(const UModelIndex &index);
```

```
    bool is_pe_index(const UModelIndex &index) { return model.rowCount(index) == 4; };
    bool is_file_index(const UModelIndex &index) {
        return model.type(index) == Types::File;
    };
    void get_unique_name(qstring &image_name);
    void get_image_guid(qstring &image_guid, UModelIndex index);
    std::vector<std::string> parseDepexSectionBody(const UModelIndex &index,
                                                   UString &parsed);
    std::vector<std::string> parseAprioriRawSection(const UModelIndex &index);
    void get_deps(UModelIndex index, std::string key);
    void get_apriori(UModelIndex index, std::string key);
    void dump_jsons(); // dump JSON with DEPEX and GUIDs information for each image
    json all_deps;     // DEPEX information for each image
    json images_guids; // matching the modules to the parent's GUIDs
    TreeModel model;
    const char *buffer;
    uint32_t buffer_size;
    std::vector<std::pair<UString, UModelIndex>> messages;
    std::set<qstring> unique_names;
    std::vector<efiloader::File *> files;
    USTATUS err;
    void set_machine_type(UByteArray pe_body);
    uint16_t machine_type = 0xffff;
    bool machine_type_detected = false;
};
} // namespace efiloader

#endif // EFILOADER_UEFITOOL_H
}
```

efiXloader/utils.cpp

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * utils.cpp
 */

#include "utils.h"

void efiloader::Utils::show_hex(void *buffer, size_t length, const char *prefix) {
    uint8_t *buf = (uint8_t *)buffer;
    msg("[efiXloader] %s = ", prefix);
    for (int i = 0; i < length; i++) {
        msg("%02x", buf[i]);
    }
```

```
    msg("\n");
}

bool efiloader::Utils::find_vol(bytevec_t &frm, std::string &sig, qoff64_t &vol_off) {
    auto found = std::search(frm.begin(), frm.end(), sig.begin(), sig.end());
    if (found != frm.end()) {
        vol_off = std::distance(frm.begin(), found);
        return true;
    } else {
        return false;
    }
}

qoff64_t efiloader::Utils::find_vol_new(linput_t *li, char *sig) {
    qoff64_t sig_off;
    char buf[5] = {0};
    while (qltell(li) != qlsize(li)) {
        qlread(li, &buf, 4);
        if (strneq(buf, sig, 4)) {
#ifdef DEBUG
            msg("[efiloader:PARSER] found FV sign %s: %#x\n", buf, qltell(li) - 4);
#endif
            return qltell(li) - 4 - 0x28;
        }
    }
    return -1;
}

qoff64_t efiloader::Utils::find_vol_test(bytevec_t &data) {
    std::string tmp(data.begin(), data.end());
    std::size_t res = tmp.find("_FVH");
    if (res != std::string::npos) {
        return res - 0x28;
    }
    return res;
}

void efiloader::Utils::skip(memory_deserializer_t *ser, size_t size, size_t count) {
    switch (size) {
    case 1:
        for (int i = 0; i < count; i++) {
            ser->unpack_db();
        }
        break;
    case 2:
        for (int i = 0; i < count; i++) {
            ser->unpack_dw();
        }
        break;
    case 4:
        for (int i = 0; i < count; i++) {
            ser->unpack_dd();
        }
        break;
    case 8:
        for (int i = 0; i < count; i++) {
            ser->unpack_dq();
        }
        break;
    default:
        break;
    }
}
}
```

efiXloader/utils.h

```
/*
 * efiXloader
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * utils.h
 */

#ifndef EFILOADER_UTILS_H
#define EFILOADER_UTILS_H

#include "ida_core.h"
#include <algorithm>
#include <cstdint>
#include <string>
#include <vector>

namespace efiloader {

class Utils {
  public:
    Utils() { ; };
    void show_hex(void *buffer, size_t length, const char *prefix);
    bool find_vol(bytevec_t &frm, std::string &sig, qoff64_t &vol_off);
    qoff64_t find_vol_new(linput_t *li, char *sig);
    qoff64_t find_vol_test(bytevec_t &data);
    void skip(memory_deserializer_t *ser, size_t size, size_t count);
};
} // namespace efiloader

#endif // EFILOADER_UTILS_H
}
```

Chapter 2.0.0

efiXplorer

efiXplorer/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.7)

project(efiXplorer CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

if(APPLE)
  # to build Mach-O universal binaries with 2 architectures
  set(CMAKE_CXX_FLAGS ${CMAKE_CXX_FLAGS} "-fPIC -arch x86_64 -arch arm64")
else()
  set(CMAKE_CXX_FLAGS ${CMAKE_CXX_FLAGS} "-fPIC")
endif()

if(BATCH)
  add_definitions(-DBATCH=1)
  set(BATCH 1)
  message(STATUS "efiXplorer plugin version for idat/idat64 binaries")
endif()

list(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/../cmake)

find_package(IdaSdk REQUIRED)

include_directories(${PROJECT_SOURCE_DIR})
include_directories(${PROJECT_SOURCE_DIR}/3rd/nlohmann_json)

# efiXplorer sources
set(efiXplorer_src
    "efiAnalyzerArm.cpp"
    "efiAnalyzerX86.cpp"
    "efiAnalyzer.h"
    "efiDeps.cpp"
    "efiDeps.h"
    "efiGlobal.h"
    "efiGlobal.cpp"
    "efiSmmUtils.cpp"
    "efiSmmUtils.h"
    "efiUi.cpp"
    "efiUi.h"
    "efiUtils.cpp"
    "efiUtils.h"
    "efiXplorer.cpp"
    "efiXplorer.h")

if(HexRaysSdk_ROOT_DIR)
  add_definitions(-DHEX_RAYS=1)
  set(HexRaysSdk_INCLUDE_DIRS ${HexRaysSdk_ROOT_DIR}/include)
  include_directories(${HexRaysSdk_INCLUDE_DIRS})
  list(APPEND efiXplorer_src "efiHexRays.cpp" "efiHexRays.h")
endif()

add_ida_plugin(efiXplorer ${PROJECT_SOURCE_DIR}/efiXplorer.cpp)

set_ida_target_properties(efiXplorer PROPERTIES CXX_STANDARD 17)
ida_target_include_directories(efiXplorer PRIVATE ${IdaSdk_INCLUDE_DIRS})

add_ida_library(efiXplorer_lib ${efiXplorer_src})
```

```
ida_target_link_libraries(efiXplorer efiXplorer_lib)
}
```

efiXplorer/efiAnalyzer.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiAnalyzer.h
 *
 */

#pragma once

#include "efiSmmUtils.h"
#include "efiUtils.h"

#ifdef HEX_RAYS
#include "efiHexRays.h"
#endif

namespace EfiAnalysis {

class EfiAnalyzer {
  public:
    EfiAnalyzer();
    ~EfiAnalyzer();

    std::vector<json> allGuids;
    std::vector<json> allProtocols;
    std::vector<json> allPPIs;
    std::vector<json> allServices;
    std::vector<func_t *> smiHandlers;
    uint8_t arch;

    void getSegments();
    void setStrings();

    void printInterfaces();
    void markInterfaces();
    void markDataGuids();

    bool efiSmmCpuProtocolResolver();
    void findSwSmiHandlers();
    bool findGetVariableOveflow(std::vector<json> allServices);
    bool findPPIGetVariableStackOveflow();
```

```
    bool findSmmGetVariableOveflow();
    bool findSmmCallout();
    bool analyzeNvramVariables();
    bool AnalyzeVariableService(ea_t ea, std::string service_str);
    bool AddProtocol(std::string serviceName, ea_t guidAddress, ea_t xrefAddress,
                     ea_t callAddress);
    void dumpInfo();

    uint8_t fileType = 0;
    json dbProtocols;
    ea_t base;
    ea_t startAddress = 0;
    ea_t endAddress = 0;
    std::vector<ea_t> funcs;
    std::filesystem::path guidsJsonPath;
    std::map<json, std::string> dbProtocolsMap; // a map to look up a GUID name
by value
    json bootServices;
    json peiServices;
    json peiServicesAll;
    json ppiCallsAll;
    json runtimeServicesAll;
    json smmServices;
    json smmServicesAll;
    std::vector<json> nvramVariables;
    std::vector<ea_t> markedInterfaces;

    // Format-dependent interface-related settings (protocols for DXE, PPIs for PEI)
    std::string if_name;
    std::string if_pl;
    std::string if_key;
    std::vector<json> *if_tbl;

    // EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID
    EfiGuid sw_guid2 = {
        0x18a3c6dc, 0x5eea, 0x48c8, {0xa1, 0xc1, 0xb5, 0x33, 0x89, 0xf9, 0x89, 0x99}};
    // EFI_SMM_SW_DISPATCH_PROTOCOL_GUID
    EfiGuid sw_guid = {
        0xe541b773, 0xdd11, 0x420c, {0xb0, 0x26, 0xdf, 0x99, 0x36, 0x53, 0xf8, 0xbf}};
    // EFI_SMM_SX_DISPATCH2_PROTOCOL_GUID
    EfiGuid sx_guid2 = {
        0x456d2859, 0xa84b, 0x4e47, {0xa2, 0xee, 0x32, 0x76, 0xd8, 0x86, 0x99, 0x7d}};
    // EFI_SMM_SX_DISPATCH_PROTOCOL_GUID
    EfiGuid sx_guid = {
        0x14FC52BE, 0x01DC, 0x426C, {0x91, 0xAE, 0xA2, 0x3C, 0x3E, 0x22, 0x0A, 0xE8}};
    // EFI_SMM_IO_TRAP_DISPATCH2_PROTOCOL_GUID
    EfiGuid io_trap_guid2 = {
        0x58DC368D, 0x7BFA, 0x4E77, {0xAB, 0xBC, 0x0E, 0x29, 0x41, 0x8D, 0xF9, 0x30}};
    // EFI_SMM_IO_TRAP_DISPATCH_PROTOCOL_GUID
    EfiGuid io_trap_guid = {
        0xDB7F536B, 0xEDE4, 0x4714, {0xA5, 0xC8, 0xE3, 0x46, 0xEB, 0xAA, 0x20, 0x1D}};
    // EFI_SMM_GPI_DISPATCH2_PROTOCOL_GUID
    EfiGuid gpi_guid2 = {
        0x25566B03, 0xB577, 0x4CBF, {0x95, 0x8C, 0xED, 0x66, 0x3E, 0xA2, 0x43, 0x80}};
    // EFI_SMM_GPI_DISPATCH_PROTOCOL_GUID
    EfiGuid gpi_guid = {
        0xE0744B81, 0x9513, 0x49CD, {0x8C, 0xEA, 0xE9, 0x24, 0x5E, 0x70, 0x39, 0xDA}};
    // EFI_SMM_USB_DISPATCH2_PROTOCOL_GUID
    EfiGuid usb_guid2 = {
        0xEE9B8D90, 0xC5A6, 0x40A2, {0xBD, 0xE2, 0x52, 0x55, 0x8D, 0x33, 0xCC, 0xA1}};
    // EFI_SMM_USB_DISPATCH_PROTOCOL_GUID
    EfiGuid usb_guid = {
        0xA05B6FFD, 0x87AF, 0x4E42, {0x95, 0xC9, 0x62, 0x28, 0xB6, 0x3C, 0xF3, 0xF3}};
    // EFI_SMM_STANDBY_BUTTON_DISPATCH2_PROTOCOL_GUID
```

```
EfiGuid standby_button_guid2 = {
    0x7300C4A1, 0x43F2, 0x4017, {0xA5, 0x1B, 0xC8, 0x1A, 0x7F, 0x40, 0x58, 0x5B}};
// EFI_SMM_STANDBY_BUTTON_DISPATCH_PROTOCOL_GUID
EfiGuid standby_button_guid = {
    0x78965B98, 0xB0BF, 0x449E, {0x8B, 0x22, 0xD2, 0x91, 0x4E, 0x49, 0x8A, 0x98}};
// EFI_SMM_PERIODIC_TIMER_DISPATCH2_PROTOCOL_GUID
EfiGuid periodic_timer_guid2 = {
    0x4CEC368E, 0x8E8E, 0x4D71, {0x8B, 0xE1, 0x95, 0x8C, 0x45, 0xFC, 0x8A, 0x53}};
// EFI_SMM_PERIODIC_TIMER_DISPATCH_PROTOCOL_GUID
EfiGuid periodic_timer_guid = {
    0x9CCA03FC, 0x4C9E, 0x4A19, {0x9B, 0x06, 0xED, 0x7B, 0x47, 0x9B, 0xDE, 0x55}};
// EFI_SMM_POWER_BUTTON_DISPATCH2_PROTOCOL_GUID
EfiGuid power_button_guid2 = {
    0x1B1183FA, 0x1823, 0x46A7, {0x88, 0x72, 0x9C, 0x57, 0x87, 0x55, 0x40, 0x9D}};
// EFI_SMM_POWER_BUTTON_DISPATCH_PROTOCOL_GUID
EfiGuid power_button_guid = {
    0xB709EFA0, 0x47A6, 0x4B41, {0xB9, 0x31, 0x12, 0xEC, 0xE7, 0xA8, 0xEE, 0x56}};
// EFI_SMM_ICHN_DISPATCH_PROTOCOL_GUID
EfiGuid ichn_guid = {
    0xC50B323E, 0x9075, 0x4F2A, {0xAC, 0x8E, 0xD2, 0x59, 0x6A, 0x10, 0x85, 0xCC}};
// EFI_SMM_ICHN_DISPATCH2_PROTOCOL_GUID
EfiGuid ichn_guid2 = {
    0xADF3A128, 0x416D, 0x4060, {0x8D, 0xDF, 0x30, 0xA1, 0xD7, 0xAA, 0xB6, 0x99}};
// PCH_TCO_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid tco_guid = {
    0x9E71D609, 0x6D24, 0x47FD, {0xB5, 0x72, 0x61, 0x40, 0xF8, 0xD9, 0xC2, 0xA4}};
// PCH_PCIE_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid pcie_guid = {
    0x3E7D2B56, 0x3F47, 0x42AA, {0x8F, 0x6B, 0x22, 0xF5, 0x19, 0x81, 0x8D, 0xAB}};
// PCH_ACPI_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid acpi_guid = {
    0xD52BB262, 0xF022, 0x49EC, {0x86, 0xD2, 0x7A, 0x29, 0x3A, 0x7A, 0x5, 0x4B}};
// PCH_GPIO_UNLOCK_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid gpio_unlock_guid = {
    0x83339EF7, 0x9392, 0x4716, {0x8D, 0x3A, 0xD1, 0xFC, 0x67, 0xCD, 0x55, 0xDB}};
// PCH_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid pch_guid = {
    0xE6A81BBF, 0x873D, 0x47FD, {0xB6, 0xBE, 0x61, 0xB3, 0xE5, 0x72, 0x9, 0x93}};
// PCH_ESPI_SMI_DISPATCH_PROTOCOL_GUID
EfiGuid espi_guid = {
    0xB3C14FF3, 0xBAE8, 0x456C, {0x86, 0x31, 0x27, 0xFE, 0x0C, 0xEB, 0x34, 0x0C}};
// EFI_ACPI_EN_DISPATCH_PROTOCOL_GUID
EfiGuid acpi_en_guid = {
    0xBD88EC68, 0xEBE4, 0x4F7B, {0x93, 0x5A, 0x4F, 0x66, 0x66, 0x42, 0xE7, 0x5F}};
// EFI_ACPI_DIS_DISPATCH_PROTOCOL_GUID
EfiGuid acpi_dis_guid = {
    0x9C939BA6, 0x1FCC, 0x46F6, {0xB4, 0xE1, 0x10, 0x2D, 0xBE, 0x18, 0x65, 0x67}};
// FCH_SMM_GPI_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_gpi_guid2 = {
    0x7051ab6d, 0x9ec2, 0x42eb, {0xa2, 0x13, 0xde, 0x48, 0x81, 0xf1, 0xf7, 0x87}};
// FCH_SMM_IO_TRAP_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_io_trap_guid2 = {
    0x91288fc4, 0xe64b, 0x4ef9, {0xa4, 0x63, 0x66, 0x88, 0x0, 0x71, 0x7f, 0xca}};
// FCH_SMM_PERIODICAL_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_periodical_guid2 = {
    0x736102f1, 0x9584, 0x44e7, {0x82, 0x8a, 0x43, 0x4b, 0x1e, 0x67, 0x5c, 0xc4}};
// FCH_SMM_PWR_BTN_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_pwr_btn_guid2 = {
    0xa365240e, 0x56b0, 0x426d, {0x83, 0xa, 0x30, 0x66, 0xc6, 0x81, 0xbe, 0x9a}};
// FCH_SMM_SW_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_sw_guid2 = {
    0x881b4ab6, 0x17b0, 0x4bdf, {0x88, 0xe2, 0xd4, 0x29, 0xda, 0x42, 0x5f, 0xfd}};
// FCH_SMM_SX_DISPATCH2_PROTOCOL_GUID
EfiGuid fch_sx_guid2 = {
```

```
        0x87e2a6cf, 0x91fb, 0x4581, {0x90, 0xa9, 0x6f, 0x50, 0x5d, 0xdc, 0x1c, 0xb2}};
    // FCH_SMM_USB_DISPATCH_PROTOCOL_GUID
    EfiGuid fch_usb_guid = {
        0x59053b0d, 0xeeb8, 0x4379, {0xb1, 0xc8, 0x14, 0x5f, 0x1b, 0xb, 0xe4, 0xb9}};
    // FCH_SMM_USB_DISPATCH2_PROTOCOL_GUID
    EfiGuid fch_usb_guid2 = {
        0xfbbb2ea9, 0xce0e, 0x4689, {0xb3, 0xf0, 0xc6, 0xb8, 0xf0, 0x76, 0xbd, 0x20}};
    // FCH_SMM_MISC_DISPATCH_PROTOCOL_GUID
    EfiGuid fch_misc_guid = {
        0x13bd659b, 0xb4c6, 0x47da, {0x9b, 0x22, 0x11, 0x50, 0xd4, 0xf3, 0xb, 0xda}};
    // FCH_SMM_APU_RAS_DISPATCH_PROTOCOL_GUID
    EfiGuid fch_apu_ras_guid = {
        0xf871ee59, 0x29d2, 0x4b15, {0x9e, 0x67, 0xaf, 0x32, 0xcd, 0xc1, 0x41, 0x73}};

    std::vector<uint64_t> ppiFlags = {
        0x1,        0x10,       0x11,       0x20,       0x21,       0x30,
        0x31,       0x40,       0x41,       0x50,       0x51,       0x60,
        0x61,       0x70,       0x71,       0x80000000, 0x80000001, 0x80000010,
        0x80000011, 0x80000020, 0x80000021, 0x80000030, 0x80000031, 0x80000040,
        0x80000041, 0x80000050, 0x80000051, 0x80000060, 0x80000061, 0x80000070,
        0x80000071,
    };

    // Set boot services that work with protocols
    std::vector<std::string> protBsNames = {"InstallProtocolInterface",
                                            "ReinstallProtocolInterface",
                                            "UninstallProtocolInterface",
                                            "HandleProtocol",
                                            "RegisterProtocolNotify",
                                            "OpenProtocol",
                                            "CloseProtocol",
                                            "OpenProtocolInformation",
                                            "ProtocolsPerHandle",
                                            "LocateHandleBuffer",
                                            "LocateProtocol",
                                            "InstallMultipleProtocolInterfaces",
                                            "UninstallMultipleProtocolInterfaces"};

    // Set smm services that work with protocols
    std::vector<std::string> protSmmNames = {"SmmInstallProtocolInterface",
                                             "SmmUninstallProtocolInterface",
                                             "SmmHandleProtocol",
                                             "SmmRegisterProtocolNotify",
                                             "SmmLocateHandle",
                                             "SmmLocateProtocol"};

    // Set of PEI services that work with PPI
    std::vector<std::string> ppiPEINames = {"InstallPpi", "ReInstallPpi", "LocatePpi",
                                            "NotifyPpi"};
};

class EfiAnalyzerX86 : public EfiAnalyzer {
  public:
    EfiAnalyzerX86() : EfiAnalyzer() {
        // import necessary types
        const til_t *idati = get_idati();
        import_type(idati, -1, "EFI_GUID");
        import_type(idati, -1, "EFI_HANDLE");
        import_type(idati, -1, "EFI_SYSTEM_TABLE");
        import_type(idati, -1, "EFI_BOOT_SERVICES");
        import_type(idati, -1, "EFI_RUNTIME_SERVICES");
        import_type(idati, -1, "_EFI_SMM_SYSTEM_TABLE2");
        import_type(idati, -1, "EFI_PEI_SERVICES");
        import_type(idati, -1, "EFI_PEI_READ_ONLY_VARIABLE2_PPI");
```

```
        import_type(idati, -1, "EFI_SMM_VARIABLE_PROTOCOL");

#ifdef HEX_RAYS
        for (auto idx = 0; idx < get_entry_qty(); idx++) {
            uval_t ord = get_entry_ordinal(idx);
            ea_t ep = get_entry(ord);
            TrackEntryParams(get_func(ep), 0);
        }
#endif
    }
    bool findImageHandleX64();
    bool findSystemTableX64();
    bool findBootServicesTables();
    bool findRuntimeServicesTables();
    bool findSmstX64();
    bool findSmstPostProcX64();
    void findOtherBsTablesX64();

    void getProtBootServicesX64();
    void getProtBootServicesX86();
    void getAllBootServices();
    void getAllRuntimeServices();
    void getAllSmmServicesX64();

    void getBsProtNamesX64();
    void getBsProtNamesX86();
    void getSmmProtNamesX64();

    void getAllPeiServicesX86();
    void getPpiNamesX86();
    void getAllVariablePPICallsX86();

    void markLocalGuidsX64();

  private:
    bool InstallMultipleProtocolInterfacesHandler();
};

class EfiAnalyzerArm : public EfiAnalyzer {
  public:
    EfiAnalyzerArm() : EfiAnalyzer() {
        // in order to make it work, it is necessary to copy
        // uefi.til, uefi64.til files in {idadir}/til/arm/
        add_til("uefi64.til", ADDTIL_DEFAULT);

        const til_t *idati = get_idati();
        import_type(idati, -1, "EFI_GUID");
        import_type(idati, -1, "EFI_HANDLE");
        import_type(idati, -1, "EFI_SYSTEM_TABLE");
        import_type(idati, -1, "EFI_BOOT_SERVICES");
        import_type(idati, -1, "EFI_RUNTIME_SERVICES");
    }
    void fixOffsets();
    void initialAnalysis();
    void findBootServicesTables();
    void initialGlobalVarsDetection();
    void servicesDetection();
    void protocolsDetection();
    void findPeiServicesFunction();

  private:
    bool getProtocol(ea_t address, uint32_t p_reg, std::string service_name);
    struct service_info_64bit {
        char service_name[64];
```

```
        uint32_t offset;
        uint32_t reg;
        uint16_t arg_index;
    };

    struct service_info_64bit bs_table_aarch64[13] = {
        {"InstallProtocolInterface", 0x80, REG_X1, 1},
        {"ReinstallProtocolInterface", 0x88, REG_X1, 1},
        {"UninstallProtocolInterface", 0x90, REG_X1, 1},
        {"HandleProtocol", 0x98, REG_X1, 1},
        {"RegisterProtocolNotify", 0xA8, REG_X0, 0},
        {"OpenProtocol", 0x118, REG_X1, 1},
        {"CloseProtocol", 0x120, REG_X1, 1},
        {"ProtocolsPerHandle", 0x128, REG_X1, 1},
        {"OpenProtocolInformation", 0x130, REG_X1, 1},
        {"LocateHandleBuffer", 0x138, REG_X1, 1},
        {"LocateProtocol", 0x140, REG_X0, 1},
        {"InstallMultipleProtocolInterfaces", 0x148, REG_X1, 1},
        {"UninstallMultipleProtocolInterfaces", 0x150, REG_X1, 1}};
};

bool efiAnalyzerMainX64();
bool efiAnalyzerMainX86();
bool efiAnalyzerMainArm();
}; // namespace EfiAnalysis

void showAllChoosers(EfiAnalysis::EfiAnalyzer analyzer);
}
```

efiXplorer/efiAnalyzerArm.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiAnalyzerArm.cpp
 * contains ARM specific analysis routines
 *
 */

#include "efiAnalyzer.h"
#include "efiGlobal.h"
#include "efiUi.h"

using namespace EfiAnalysis;

std::vector<ea_t> gImageHandleListArm;
```

```
std::vector<ea_t> gStListArm;
std::vector<ea_t> gBsListArm;
std::vector<ea_t> gRtListArm;

void EfiAnalysis::EfiAnalyzerArm::fixOffsets() {
    insn_t insn;
    for (auto func_addr : funcs) {
        func_t *f = get_func(func_addr);
        if (f == nullptr) {
            continue;
        }
        ea_t ea = f->start_ea;
        while (ea < f->end_ea) {
            ea = next_head(ea, BADADDR);
            decode_insn(&insn, ea);
            if (insn.itype == ARM_str) {
                continue;
            }
            if (insn.ops[0].type == o_displ) {
                op_num(ea, 0);
            }
            if (insn.ops[1].type == o_displ) {
                op_num(ea, 1);
            }
        }
    }
}

void EfiAnalysis::EfiAnalyzerArm::initialAnalysis() {
    fixOffsets();
    for (auto idx = 0; idx < get_entry_qty(); idx++) {
        uval_t ord = get_entry_ordinal(idx);
        ea_t ep = get_entry(ord);
        set_name(ep, "_ModuleEntryPoint", SN_FORCE);
#ifdef HEX_RAYS
        TrackEntryParams(get_func(ep), 0);
#endif /* HEX_RAYS */
    }
    if (fileType == FTYPE_PEI) {
        // setEntryArgToPeiSvc();
    }
}

ea_t getTable(ea_t code_addr, uint64_t offset) {
    ea_t table = BADADDR;
    insn_t insn;
    decode_insn(&insn, code_addr);
    if (insn.itype != ARM_ldr || insn.ops[0].type != o_reg ||
        insn.ops[1].type != o_displ || insn.ops[1].addr != offset ||
        insn.ops[1].reg == REG_XSP) {
        return table;
    }
    uint8_t table_reg = insn.ops[0].reg;
    uint8_t st_reg = insn.ops[1].reg;

    // handle following code patterns
    // ADR    REG1, gBS
    // LDR    REG2, [REG3,#0x60] <- we are here
    // STR    REG2, [REG4]
    ea_t ea = code_addr;
    uint8_t adr_reg = 0xff;
    while (true) {
        ea = next_head(ea, BADADDR);
        decode_insn(&insn, ea);
```

```
        if (insn.itype == ARM_str && insn.ops[0].type == o_reg &&
            insn.ops[1].type == o_displ && insn.ops[1].addr == 0x0) {
            adr_reg = insn.ops[1].reg;
        }
        if (is_basic_block_end(insn, false)) {
            break;
        }
    }
    if (adr_reg != 0xff) {
        ea = code_addr;
        while (true) {
            ea = prev_head(ea, 0);
            decode_insn(&insn, ea);
            if (insn.itype == ARM_adr && insn.ops[0].type == o_reg &&
                insn.ops[0].reg == adr_reg && insn.ops[1].type == o_imm) {
                return insn.ops[1].value; // gBS/gRT
            }
            if (is_basic_block_end(insn, false)) {
                break;
            }
        }
    }

    ea = code_addr;
    while (true) {
        ea = next_head(ea, BADADDR);
        decode_insn(&insn, ea);
        ea_t base = BADADDR;
        uint8_t reg = 0xff;
        if (insn.itype == ARM_adrp && insn.ops[0].type == o_reg &&
            insn.ops[1].type == o_imm) {
            // Example:
            // LDR   X8, [X1,#0x58]
            // ADRP  X9, #gRT@PAGE <- we are here
            // ...
            // STR   X8, [X9,#gRT@PAGEOFF]

            base = insn.ops[1].value;
            reg = insn.ops[0].reg;
            ea_t current_addr = ea;
            while (true) {
                current_addr = next_head(current_addr, BADADDR);
                decode_insn(&insn, current_addr);
                if (insn.itype == ARM_str && insn.ops[0].type == o_reg &&
                    insn.ops[0].reg == table_reg && insn.ops[1].type == o_displ &&
                    insn.ops[1].reg == reg) {
                    return base + insn.ops[1].addr;
                }
                if (is_basic_block_end(insn, false)) {
                    break;
                }
            }
        }
        if (is_basic_block_end(insn, false)) {
            break;
        }
    }
    return table;
}

json getService(ea_t addr, uint8_t table_id) {
    json s;
    insn_t insn;
    decode_insn(&insn, addr);
```

```
    if (insn.itype == ARM_ldr && insn.ops[0].type == o_reg &&
        insn.ops[1].type == o_displ) {
        ea_t ea = addr;
        uint8_t blr_reg = 0xff;
        uint8_t table_reg = insn.ops[0].reg;
        uint64_t service_offset = BADADDR;
        while (true) {
            ea = next_head(ea, BADADDR);
            decode_insn(&insn, ea);
            if (insn.itype == ARM_ldr && insn.ops[0].type == o_reg &&
                insn.ops[1].type == o_displ && insn.ops[1].reg == table_reg) {
                service_offset = insn.ops[1].addr;
                blr_reg = insn.ops[0].reg;
            }
            if (blr_reg != 0xff && service_offset != BADADDR && insn.itype ==
ARM_blr &&
                insn.ops[0].type == o_reg && insn.ops[0].reg == blr_reg) {
                s["address"] = ea;
                if (table_id == 1) {
                    s["service_name"] = lookupBootServiceName(service_offset);
                    s["table_name"] = "EFI_BOOT_SERVICES";
                } else if (table_id == 2) {
                    s["service_name"] = lookupRuntimeServiceName(service_offset);
                    s["table_name"] = "EFI_RUNTIME_SERVICES";
                } else {
                    s["table_name"] = "OTHER";
                }
                return s;
            }
            if (is_basic_block_end(insn, false)) {
                break;
            }
        }
    }

    // handle following code patterns
    // ADR    REG1, gBS
    // ...
    // LDR    REG2, [REG1]
    // ...
    // LDR    REG3, [REG2,#0x28]
    if (insn.itype == ARM_adr && insn.ops[0].type == o_reg && insn.ops[1].type ==
o_imm) {
        uint8_t reg1 = insn.ops[0].reg;
        uint8_t reg2 = 0xff;
        ea_t ea = addr;
        while (true) {
            ea = next_head(ea, BADADDR);
            decode_insn(&insn, ea);
            if (insn.itype == ARM_ldr && insn.ops[0].type == o_reg &&
                insn.ops[1].type == o_displ && insn.ops[1].reg == reg1 &&
                insn.ops[1].addr == 0) {
                reg2 = insn.ops[0].reg;
            }
            if (reg2 != 0xff && insn.itype == ARM_ldr && insn.ops[0].type == o_reg &&
                insn.ops[1].type == o_displ && insn.ops[1].reg == reg2) {
                s["address"] = ea;
                if (table_id == 1) {
                    s["service_name"] = lookupBootServiceName(insn.ops[1].addr);
                    s["table_name"] = "EFI_BOOT_SERVICES";
                } else if (table_id == 2) {
                    s["service_name"] = lookupRuntimeServiceName(insn.ops[1].addr);
                    s["table_name"] = "EFI_RUNTIME_SERVICES";
                } else {
```

```
                    s["table_name"] = "OTHER";
                }
                return s;
            }
        }
    }

    return s;
}

void EfiAnalysis::EfiAnalyzerArm::initialGlobalVarsDetection() {
#ifdef HEX_RAYS
    // analyze entry point with Hex-Rays
    for (auto func_addr : funcs) {
        json res = DetectVars(get_func(func_addr));
        if (res.contains("gImageHandleList")) {
            for (auto addr : res["gImageHandleList"]) {
                if (!addrInVec(gImageHandleListArm, addr)) {
                    gImageHandleListArm.push_back(addr);
                }
            }
        }
        if (res.contains("gStList")) {
            for (auto addr : res["gStList"]) {
                if (!addrInVec(gStListArm, addr)) {
                    gStListArm.push_back(addr);
                }
            }
        }
        if (res.contains("gBsList")) {
            for (auto addr : res["gBsList"]) {
                if (!addrInVec(gBsListArm, addr)) {
                    gBsListArm.push_back(addr);
                }
            }
        }
        if (res.contains("gRtList")) {
            for (auto addr : res["gRtList"]) {
                if (!addrInVec(gRtListArm, addr)) {
                    gRtListArm.push_back(addr);
                }
            }
        }
    }
#endif /* HEX_RAYS */

    // analysis of all functions and search for additional table initializations
    for (auto func_addr : funcs) {
        func_t *f = get_func(func_addr);
        if (f == nullptr) {
            continue;
        }
        auto ea = f->start_ea;
        while (ea < f->end_ea) {
            ea = next_head(ea, BADADDR);
            ea_t bs = getTable(ea, 0x60);
            if (bs != BADADDR) {
                msg("[efiXplorer] gBS = 0x%016llX\n", u64_addr(ea));
                setPtrTypeAndName(bs, "gBS", "EFI_BOOT_SERVICES");
                if (!addrInVec(gBsListArm, bs)) {
                    gBsListArm.push_back(bs);
                }
                continue;
            }
```

```
            ea_t rt = getTable(ea, 0x58);
            if (rt != BADADDR) {
                msg("[efiXplorer] gRT = 0x%016llX\n", u64_addr(ea));
                setPtrTypeAndName(rt, "gRT", "EFI_RUNTIME_SERVICES");
                if (!addrInVec(gRtListArm, rt)) {
                    gRtListArm.push_back(rt);
                }
                continue;
            }
        }
    }
}

void EfiAnalysis::EfiAnalyzerArm::servicesDetection() {

#ifdef HEX_RAYS
    for (auto func_addr : funcs) {
        std::vector<json> services = DetectServices(get_func(func_addr));
        for (auto service : services) {
            allServices.push_back(service);
        }
    }
#endif /* HEX_RAYS */

    // analyze xrefs to gBS, gRT
    for (auto bs : gBsListArm) {
        auto xrefs = getXrefs(bs);
        for (auto ea : xrefs) {
            auto s = getService(ea, 1);
            if (!s.contains("address")) {
                continue;
            }
            std::string name = s["service_name"];
            if (name == "Unknown") {
                continue;
            }
            if (!jsonInVec(allServices, s)) {
                msg("[efiXplorer] gBS xref address: 0x%016llX, found new service\n",
                    u64_addr(ea));
                allServices.push_back(s);
            }
        }
    }
    for (auto rt : gRtListArm) {
        auto xrefs = getXrefs(rt);
        for (auto ea : xrefs) {
            auto s = getService(ea, 2);
            if (!s.contains("address")) {
                continue;
            }
            std::string name = s["service_name"];
            if (name == "Unknown") {
                continue;
            }
            if (!jsonInVec(allServices, s)) {
                msg("[efiXplorer] gRT xref address: 0x%016llX, found new service\n",
                    u64_addr(ea));
                allServices.push_back(s);
            }
        }
    }
}

bool EfiAnalysis::EfiAnalyzerArm::getProtocol(ea_t address, uint32_t p_reg,
```

```
                                                std::string service_name) {
    ea_t ea = address;
    insn_t insn;
    ea_t offset = BADADDR;
    ea_t guid_addr = BADADDR;
    ea_t code_addr = BADADDR;
    while (true) {
        ea = prev_head(ea, 0);
        decode_insn(&insn, ea);
        if (insn.itype == ARM_adrl && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == p_reg && insn.ops[1].type == o_imm) {
            guid_addr = insn.ops[1].value;
            code_addr = ea;
            break;
        }
        if (insn.itype == ARM_add && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == p_reg && insn.ops[1].type == o_reg &&
            insn.ops[1].reg == p_reg && insn.ops[2].type == o_imm) {
            offset = insn.ops[2].value;
        }
        if (insn.itype == ARM_adrp && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == p_reg && insn.ops[1].type == o_imm) {
            guid_addr = insn.ops[1].value + offset;
            code_addr = ea;
            break;
        }
        if (is_basic_block_end(insn, false)) {
            break;
        }
    }
    if (guid_addr == BADADDR || code_addr == BADADDR) {
        return false;
    }
    msg("[efiXplorer] address: 0x%016llX, found new protocol\n", u64_addr(code_addr));
    return AddProtocol(service_name, guid_addr, code_addr, address);
}

void EfiAnalysis::EfiAnalyzerArm::protocolsDetection() {
    for (auto s : allServices) {
        std::string service_name = s["service_name"];
        for (auto i = 0; i < 13; i++) {
            std::string current_name =
                static_cast<std::string>(bs_table_aarch64[i].service_name);
            if (current_name != service_name) {
                continue;
            }
            getProtocol(s["address"], bs_table_aarch64[i].reg, service_name);
            break;
        }
    }
}

void EfiAnalysis::EfiAnalyzerArm::findPeiServicesFunction() {
    insn_t insn;
    for (auto start_ea : funcs) {
        decode_insn(&insn, start_ea);
        if (!(insn.itype == ARM_mrs && insn.ops[0].type == o_reg &&
              insn.ops[0].reg == REG_X0 && insn.ops[1].type == o_imm &&
              insn.ops[1].value == 0x3 && insn.ops[2].type == o_idpspec3 &&
              insn.ops[2].reg == REG_C13 && insn.ops[3].type == o_idpspec3 &&
              insn.ops[3].reg == REG_C0 && insn.ops[4].type == o_imm &&
              insn.ops[4].value == 0x2)) {
            continue;
        }
```

```
        auto end_ea = next_head(start_ea, BADADDR);
        if (end_ea == BADADDR) {
            continue;
        }
        decode_insn(&insn, end_ea);
        if (insn.itype == ARM_ret) {
            msg("[efiXplorer] found GetPeiServices() function: 0x%016llX\n",
                u64_addr(start_ea));
            set_name(start_ea, "GetPeiServices", SN_FORCE);
            setRetToPeiSvc(start_ea);
        }
    }
}

//--------------------------------------------------------------------------
// Show all non-empty choosers windows
void showAllChoosers(EfiAnalysis::EfiAnalyzerArm analyzer) {
    qstring title;

    // open window with all services
    if (analyzer.allServices.size()) {
        title = "efiXplorer: services";
        services_show(analyzer.allServices, title);
    }

    // open window with data guids
    if (analyzer.allGuids.size()) {
        qstring title = "efiXplorer: GUIDs";
        guids_show(analyzer.allGuids, title);
    }

    // open window with protocols
    if (analyzer.allProtocols.size()) {
        title = "efiXplorer: protocols";
        protocols_show(analyzer.allProtocols, title);
    }
}

//--------------------------------------------------------------------------
// Main function for AARCH64 modules
bool EfiAnalysis::efiAnalyzerMainArm() {

    show_wait_box("HIDECANCEL\nAnalyzing module(s) with efiXplorer...");

    EfiAnalysis::EfiAnalyzerArm analyzer;

    while (!auto_is_ok()) {
        auto_wait();
    };

    // find .text and .data segments
    analyzer.getSegments();

    // mark GUIDs
    analyzer.markDataGuids();

    if (g_args.disable_ui) {
        analyzer.fileType = g_args.module_type == PEI
                                ? analyzer.fileType = FTYPE_PEI
                                : analyzer.fileType = FTYPE_DXE_AND_THE_LIKE;
    } else {
        analyzer.fileType = getFileType(&analyzer.allGuids);
    }
```

```
    if (analyzer.fileType == FTYPE_PEI) {
        msg("[efiXplorer] input file is PEI module\n");
    }

    // set the correct name for the entry point and automatically fix the prototype
    analyzer.initialAnalysis();

    if (analyzer.fileType == FTYPE_DXE_AND_THE_LIKE) {
        analyzer.initialGlobalVarsDetection();

        // detect services
        analyzer.servicesDetection();

        // detect protocols
        analyzer.protocolsDetection();
    } else if (analyzer.fileType == FTYPE_PEI) {
        analyzer.findPeiServicesFunction();
    }

#ifdef HEX_RAYS
    for (auto addr : analyzer.funcs) {
        std::vector<json> services = DetectPeiServicesArm(get_func(addr));
        for (auto service : services) {
            analyzer.allServices.push_back(service);
        }
    }
    applyAllTypesForInterfacesBootServices(analyzer.allProtocols);
#endif /* HEX_RAYS */
    showAllChoosers(analyzer);

    analyzer.dumpInfo();

    hide_wait_box();

    return true;
}
}
```

efiXplorer/efiDeps.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiDeps.cpp
 *
 */
```

```
#include "efiDeps.h"

EfiDependencies::EfiDependencies() {
    // Read DEPEX (for protocols) from
    // .deps.json file if this file exists
    loadDepsFromUefiTool();
    // Get images names from IDB
    getImages();
    // Read images with GUIDs from
    // .images.json file if this file exists
    loadImagesWithGuids();
};

EfiDependencies::~EfiDependencies() {
    imagesInfo.clear();
    imagesGuids.clear();
    imagesFromIdb.clear();
    uefitoolDeps.clear();
    modulesSequence.clear();
    protocolsChooser.clear();
    protocolsByGuids.clear();
    additionalInstallers.clear();
    protocolsWithoutInstallers.clear();
};

json EfiDependencies::getDeps(std::string guid) {
    json res;
    std::vector installers({"InstallProtocolInterface",
                            "InstallMultipleProtocolInterfaces",
                            "SmmInstallProtocolInterface"});
    for (auto &it : protocolsChooser.items()) {
        auto p = it.value();
        if (p["guid"] != guid) {
            continue;
        }
        p["ea"] = getHex(u64_addr(p["ea"]));
        p["xref"] = getHex(u64_addr(p["xref"]));
        p["address"] = getHex(u64_addr(p["address"]));
        if (find(installers.begin(), installers.end(), p["service"]) !=
            installers.end()) {
            res["installed"].push_back(p);
        } else {
            res["used"].push_back(p);
        }
    }

    return res;
}

void EfiDependencies::getProtocolsByGuids(std::vector<json> protocols) {
    for (auto p : protocols) {
        // check if entry for GUID already exist
        std::string guid = p["guid"];
        auto deps = protocolsByGuids[guid];
        if (deps.is_null()) {
            protocolsByGuids[guid] = getDeps(guid);
        }
    }
}

void EfiDependencies::getProtocolsChooser(std::vector<json> protocols) {
    auto i = 0;
    for (auto p : protocols) {
```

```
        protocolsChooser[i] = p;
        ++i;
    }
}

bool EfiDependencies::loadDepsFromUefiTool() {
    std::filesystem::path deps_json;
    deps_json /= get_path(PATH_TYPE_IDB);
    deps_json.replace_extension(".deps.json");
    if (!std::filesystem::exists(deps_json)) {
        return false;
    }
    std::ifstream file(deps_json);
    file >> uefitoolDeps;
    return true;
}

bool EfiDependencies::loadImagesWithGuids() {
    std::filesystem::path images_json;
    images_json /= get_path(PATH_TYPE_IDB);
    images_json.replace_extension(".images.json");
    if (!std::filesystem::exists(images_json)) {
        return false;
    }
    std::ifstream file(images_json);
    file >> imagesGuids;
    return true;
}

bool EfiDependencies::installerFound(std::string protocol) {
    auto deps_prot = protocolsByGuids[protocol];
    if (deps_prot.is_null()) {
        return false;
    }
    auto installers = deps_prot["installed"];
    if (installers.is_null()) {
        return false;
    }
    return true;
}

void EfiDependencies::getProtocolsWithoutInstallers() {
    // Check DXE_DEPEX and MM_DEPEX
    std::vector<std::string> sections{"EFI_SECTION_DXE_DEPEX", "EFI_SECTION_MM_DEPEX"};
    for (auto section : sections) {
        auto images = uefitoolDeps[section];
        for (auto &element : images.items()) {
            auto protocols = element.value();
            for (auto p : protocols) {
                std::string ps = static_cast<std::string>(p);
                if (!installerFound(ps)) {
                    protocolsWithoutInstallers.insert(ps);
                }
            }
        }
    }
}

void EfiDependencies::getInstallersModules() {
    // search for this protocols in binary
    for (auto &protocol : protocolsWithoutInstallers) {
        auto addrs = searchProtocol(protocol);
        bool installerFound = false;
        for (auto addr : addrs) {
```

```
            auto xrefs = getXrefs(addr);
            if (!xrefs.size()) {
                continue;
            }
            if (xrefs.size() == 1) {
                func_t *func = get_func(xrefs.at(0));
                if (func == nullptr) {
                    xrefs = getXrefsToArray(xrefs.at(0));
                }
            }
            for (auto ea : xrefs) {
                if (checkInstallProtocol(ea)) {
                    auto module = getModuleNameLoader(ea);
                    additionalInstallers[protocol] =
                        static_cast<std::string>(module.c_str());
                    installerFound = true;
                    break;
                }
            }
            if (installerFound) {
                break;
            }
        }
        if (!installerFound) {
            untrackedProtocols.insert(protocol);
        }
    }
}

void EfiDependencies::getAdditionalInstallers() {
    getProtocolsWithoutInstallers();
    getInstallersModules();
    std::string installers = additionalInstallers.dump(2);
    msg("Additional installers: %s\n", installers.c_str());
    msg("Untracked protocols:\n");
    for (auto &protocol : untrackedProtocols) {
        msg("%s\n", protocol.c_str());
    }
}

void EfiDependencies::getImages() {
    for (segment_t *s = get_first_seg(); s != nullptr; s = get_next_seg(s->start_ea)) {
        qstring seg_name;
        get_segm_name(&seg_name, s);

        std::vector<std::string> codeSegNames{"_.text", "_.code"};
        for (auto name : codeSegNames) {
            auto index = seg_name.find(name.c_str());
            if (index != std::string::npos) {
                std::string image_name =
                    static_cast<std::string>(seg_name.c_str()).substr(0, index);
                if (!image_name.rfind("_", 0)) {
                    image_name = image_name.erase(0, 1);
                }
                imagesFromIdb.push_back(image_name);
            }
        }
    }
}

json EfiDependencies::getImageInfo(std::string image) {
    json info;
    std::vector<std::string> installedProtocols;
    json depsProtocols;
```

```
    std::vector installers({"InstallProtocolInterface",
                            "InstallMultipleProtocolInterfaces",
                            "SmmInstallProtocolInterface"});

    // Get installed protocols
    for (auto &p : additionalInstallers.items()) { // check additional installers
        std::string adInstImage = p.value();
        std::string adInstProtocol = p.key();
        if (adInstImage == image) {
            installedProtocols.push_back(adInstProtocol);
            break;
        }
    }

    for (auto &element : protocolsChooser.items()) { // check efiXplorer report
        json p = element.value();
        std::string image_name = p["module"];
        if (!image_name.rfind("_", 0)) {
            image_name = image_name.erase(0, 1);
        }
        if (image_name != image) {
            continue;
        }
        if (find(installers.begin(), installers.end(), p["service"]) !=
            installers.end()) {
            installedProtocols.push_back(p["guid"]);
        }
    }

    // Get deps
    bool found = false;
    std::vector<std::string> sections{"EFI_SECTION_DXE_DEPEX", "EFI_SECTION_MM_DEPEX"};
    for (auto section : sections) {
        json deps_images = uefitoolDeps[section];
        for (auto &element : deps_images.items()) {
            std::string dimage_guid = element.key();
            if (imagesGuids[dimage_guid].is_null()) {
                // Can not get name for image
                continue;
            }
            std::string dimage_name = imagesGuids[dimage_guid];
            if (dimage_name == image) {
                depsProtocols = element.value();
                found = true;
                break;
            }
        }
        if (found) {
            break;
        }
    }

    info["installed_protocols"] = installedProtocols;
    info["deps_protocols"] = depsProtocols;

    return info;
}

bool EfiDependencies::getImagesInfo() {
    if (imagesInfo.size()) {
        return true;
    }
    for (auto image : imagesFromIdb) {
        imagesInfo[image] = getImageInfo(image);
```

```
    }
    return true;
}

std::string EfiDependencies::getInstaller(std::string protocol) {
    std::string res;
    for (auto &e : imagesInfo.items()) {
        std::string image = e.key();
        std::vector<std::string> installers = imagesInfo[image]["installed_protocols"];
        if (find(installers.begin(), installers.end(), protocol) != installers.end()) {
            return image;
        }
    }
    return res;
}

bool EfiDependencies::buildModulesSequence() {
    if (modulesSequence.size()) {
        return true;
    }

    std::set<std::string> modulesSeq;
    std::set<std::string> installed_protocols;

    getProtocolsWithoutInstallers(); // hard to find installers for all protocols in
                                     // static
    getImagesInfo();

    size_t index = 0;
    while (modulesSeq.size() != imagesInfo.size()) {
        bool changed = false;
        for (auto &e : imagesInfo.items()) {
            std::string image = e.key(); // current module

            // check if the image is already loaded
            if (modulesSeq.find(image) != modulesSeq.end()) {
                continue;
            }

            std::vector<std::string> installers =
                imagesInfo[image]["installed_protocols"];

            // if there are no dependencies
            if (imagesInfo[image]["deps_protocols"].is_null()) {
                for (auto protocol : installers) {
                    installed_protocols.insert(protocol);
                }
                modulesSeq.insert(image);
                json info;
                info["module"] = image;
                modulesSequence[index++] = info;
                changed = true;
                continue;
            }

            std::vector<std::string> deps = imagesInfo[image]["deps_protocols"];
            std::vector<std::string> unresolved_deps;
            bool load = true;
            for (auto protocol : deps) {
                if (installed_protocols.find(protocol) != installed_protocols.end()) {
                    continue;
                }
                if (protocolsWithoutInstallers.find(protocol) !=
                    protocolsWithoutInstallers.end()) {
```

```
                    unresolved_deps.push_back(protocol);
                    continue;
                }
                load = false;
                break;
            }

            if (load) {
                for (auto protocol : installers) {
                    installed_protocols.insert(protocol);
                }
                modulesSeq.insert(image);
                json info;
                info["image"] = image;
                info["deps"] = deps;
                if (unresolved_deps.size()) {
                    info["unresolved_deps"] = unresolved_deps;
                }
                modulesSequence[index++] = info;
                changed = true;
            }
        }

        if (!changed) { // we are in a loop, we need to load a module that installs the
                        // most popular protocol
            std::map<std::string, size_t>
                protocols_usage; // get the most popular protocol
            for (auto &e : imagesInfo.items()) {
                std::string image = e.key();

                // check if the image is already loaded
                if (modulesSeq.find(image) != modulesSeq.end()) {
                    continue;
                }

                if (imagesInfo[image]["deps_protocols"].is_null()) {
                    continue;
                }

                std::vector<std::string> deps_protocols =
                    imagesInfo[image]["deps_protocols"];
                for (auto protocol : deps_protocols) {
                    if (installed_protocols.find(protocol) !=
installed_protocols.end()) {
                        continue;
                    }
                    if (protocolsWithoutInstallers.find(protocol) !=
                        protocolsWithoutInstallers.end()) {
                        continue;
                    }
                    if (protocols_usage.find(protocol) == protocols_usage.end()) {
                        protocols_usage[protocol] = 1;
                    } else {
                        protocols_usage[protocol] += 1;
                    }
                }
            }
            std::string mprotocol;
            size_t mnum = 0;
            for (auto const &[prot, counter] : protocols_usage) {
                if (counter > mnum) {
                    mnum = static_cast<size_t>(counter);
                    mprotocol = static_cast<std::string>(prot);
                }
```

```
            }
            if (!mnum) {
                break; // the most popular protocol was not found
            }
            // find installer module for mprotocol
            std::string installer_image = getInstaller(mprotocol);
            if (!installer_image.size()) {
                msg("Can not find installer for protocol %s\n", mprotocol.c_str());
                break; // something went wrong, extra mitigation for an infinite loop
            }
            // load installer_image
            std::vector<std::string> current_installers =
                imagesInfo[installer_image]["installed_protocols"];
            for (auto protocol : current_installers) {
                installed_protocols.insert(protocol);
            }
            modulesSeq.insert(installer_image);
            json info;
            info["image"] = installer_image;
            info["deps"] = imagesInfo[installer_image]["deps_protocols"];
            modulesSequence[index++] = info;
        }
    }

    return true;
}
}
```

efiXplorer/efiDeps.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiDeps.h
 *
 */

#pragma once

#include "efiUtils.h"

class EfiDependencies {
  public:
    EfiDependencies();
    ~EfiDependencies();
    json protocolsByGuids; // protocols sorted by GUIDs
```

```
    json protocolsChooser; // numbered json with protocols
    json uefitoolDeps;
    json imagesGuids;
    json additionalInstallers; // getAdditionalInstallers result
    json imagesInfo;           // getImagesInfo result
    json modulesSequence;      // buildModulesSequence result
    std::vector<std::string> imagesFromIdb;
    std::set<std::string> untrackedProtocols;
    // Input: protocols from report
    void getProtocolsByGuids(std::vector<json> protocols);
    void getProtocolsChooser(std::vector<json> protocols);
    json getDeps(std::string protocol); // get dependencies for specific protocol
    void getAdditionalInstallers(); // get installers by protocol GUIDs by searching in
                                    // the firmware and analyzing xrefs
    bool buildModulesSequence();
    bool getImagesInfo();

  private:
    void getImages();
    std::set<std::string> protocolsWithoutInstallers;
    void getProtocolsWithoutInstallers();
    void getInstallersModules();
    bool loadDepsFromUefiTool();
    bool loadImagesWithGuids();
    bool installerFound(std::string protocol);
    json getImageInfo(std::string image);
    std::string getInstaller(std::string protocol);
};
}
```

efiXplorer/efiGlobal.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiGlobal.cpp
 *
 */

#include "efiGlobal.h"

EfiDependencies g_deps;
}
```

efiXplorer/efiGlobal.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiGlobal.h
 *
 */

#include "efiDeps.h"

enum module_types { DXE_SMM = 0, PEI = 1 };

struct args {
    int module_type;
    int disable_ui;
    int disable_vuln_hunt;
};

extern struct args g_args;
extern EfiDependencies g_deps;
}
```

efiXplorer/efiHexRays.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly, Rolf Rolles
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiHexRays.cpp
```

```
 *
 */

#include "efiHexRays.h"

// Given a tinfo_t specifying a user-defined type (UDT), look up the specified
// field by its name, and retrieve its offset.
bool offsetOf(tinfo_t tif, const char *name, unsigned int *offset) {
    // Get the udt details
    udt_type_data_t udt;
    if (!tif.get_udt_details(&udt)) {
        qstring str;
        tif.get_type_name(&str);
        return false;
    }

    // Find the udt member
    udt_member_t udm;
    udm.name = name;
    int fIdx = tif.find_udt_member(&udm, STRMEM_NAME);
    if (fIdx < 0) {
        qstring tstr;
        tif.get_type_name(&tstr);
        return false;
    }

    // Get the offset of the field
    *offset = static_cast<unsigned int>(udt.at(fIdx).offset >> 3ULL);
    return true;
}

// Utility function to set a Hex-Rays variable type and set types for the interfaces
bool setHexRaysVariableInfoAndHandleInterfaces(ea_t funcEa, lvar_t &ll, tinfo_t tif,
                                               std::string name) {
    lvar_saved_info_t lsi;
    lsi.ll = ll;
    lsi.type = tif;
    modify_user_lvar_info(funcEa, MLI_TYPE, lsi);

    // Set lvar name
    if (ll.is_stk_var()) { // Rename local variable on stack
        sval_t stkoff = ll.get_stkoff();
        struc_t *frame = get_frame(funcEa);
        set_member_name(frame, stkoff, name.c_str());
    } else { // Modufy user lvar info
        lsi.name = static_cast<qstring>(name.c_str());
        modify_user_lvar_info(funcEa, MLI_NAME, lsi);
    }

    // Get xrefs to local variable
    xreflist_t xrefs = xrefsToStackVar(funcEa, static_cast<qstring>(name.c_str()));
    qstring typeName;
    ptr_type_data_t pi;
    tif.get_ptr_details(&pi);
    pi.obj_type.get_type_name(&typeName);
    // Handling all interface functions (to rename function arguments)
    opstroffForInterface(xrefs, typeName);

    return true;
}

// Utility function to set a Hex-Rays variable name
bool setLvarName(qstring name, lvar_t lvar, ea_t func_addr) {
    lvar_saved_info_t lsi;
```

```
    lvar_uservec_t lvuv;

    lsi.ll = lvar;
    lsi.name = name;
    if (!lvuv.lvvec.add_unique(lsi)) {
        return false;
    }
    save_user_lvar_settings(func_addr, lvuv);
    return true;
}

// Utility function to set a Hex-Rays variable type and name
bool setHexRaysVariableInfo(ea_t funcEa, lvar_t &ll, tinfo_t tif, std::string name) {
    lvar_saved_info_t lsi;
    lsi.ll = ll;
    lsi.type = tif;
    modify_user_lvar_info(funcEa, MLI_TYPE, lsi);

    // Set lvar name
    if (ll.is_stk_var()) { // Rename local variable on stack
        sval_t stkoff = ll.get_stkoff();
        struc_t *frame = get_frame(funcEa);
        set_member_name(frame, stkoff, name.c_str());
    } else { // Modufy user lvar info
        lsi.name = static_cast<qstring>(name.c_str());
        modify_user_lvar_info(funcEa, MLI_NAME, lsi);
    }

    return true;
}

// I added this bit of logic when I noticed that sometimes Hex-Rays will
// aggressively create arrays on the stack. So, I wanted to apply types to
// stack "variables" (whose pointers are passed to the protocol location
// functions), but according to Hex-Rays, they weren't "variables", they
// were arrays. This bit of logic generically detects arrays of either POD
// types, or perhaps pointers to POD types. The final argument allows the
// caller to specify the maximum depth "depth" of the pointers. E.g. at
// depth 1, "int *[10]" is acceptable. At depth 2, "int **[10]" is acceptable.
bool isPODArray(tinfo_t tif, unsigned int ptrDepth = 0) {
    // If it's not an array, we're done
    if (!tif.is_array())
        return false;

    qstring tstr;

    // If it is an array, we should be able to get its array details.
    array_type_data_t atd;
    if (!tif.get_array_details(&atd)) {
        tif.get_type_name(&tstr);
        return false;
    }

    // Get the element type from the array
    tinfo_t et = atd.elem_type;

    // Start off with depth + 1, so the loop will execute at least once
    int iDepth = ptrDepth + 1;

    // Loop over the user-specified depth
    while (iDepth > 0) {

        // Use typeid last checks. I should clean this up; I'm sure I can get rid
        // of one of them.
```

```
        bool b1 = is_typeid_last(et.get_realtype());
        bool b2 = et.is_decl_last();

        // Debug printing
        et.get_type_name(&tstr);

        // If it was an integer type, return true
        if (b1 || b2)
            return true;

        // Otherwise, this is where the "pointer depth" comes in.
        // If we haven't exhausted the pointer depth,
        if (--iDepth > 0) {
            // Remove one layer of indirection from the element type
            if (et.is_ptr())
                et = remove_pointer(et);

            // Unless it's not a pointer, then return false.
            else
                return false;
        }
    }

    // If the array wasn't pointers of POD types up to the specified depth, we
    // failed. Return false.
    return false;
}

// Utility function to get a printable qstring from a cexpr_t
const char *Expr2String(cexpr_t *e, qstring *out) {
    e->print1(out, NULL);
    tag_remove(out);
    return out->c_str();
}

bool applyAllTypesForInterfacesBootServices(std::vector<json> protocols) {
    if (!init_hexrays_plugin()) {
        return false;
    }

    // Descriptors for EFI_BOOT_SERVICES functions
    struct TargetFunctionPointer BootServicesFunctions[5]{
        {"InstallProtocolInterface", 0x80, 4, 1, 3},
        {"HandleProtocol", 0x98, 3, 1, 2},
        {"OpenProtocol", 0x118, 6, 1, 2},
        {"LocateProtocol", 0x140, 3, 0, 2},
        {"InstallMultipleProtocolInterfaces", 0x148, 4, 1, 2}};

    // Initialize
    ServiceDescriptor sdBs;
    sdBs.Initialize("EFI_BOOT_SERVICES", BootServicesFunctions, 5);

    ServiceDescriptorMap mBs;
    mBs.Register(sdBs);

    GUIDRetyper retyperBs(mBs);
    retyperBs.SetProtocols(protocols);

    // Handle all protocols
    for (auto protocol : protocols) {
        auto code_addr = protocol["ea"];
        auto service = protocol["service"];

        func_t *f = get_func(code_addr);
```

```
        if (f == nullptr) {
            continue;
        }

        retyperBs.SetCodeEa(code_addr);
        retyperBs.SetFuncEa(f->start_ea);

        hexrays_failure_t hf;
        cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);

        // Check that the function is decompiled
        if (cfunc == nullptr) {
            continue;
        }

        retyperBs.apply_to(&cfunc->body, nullptr);
    }

    return true;
}

bool applyAllTypesForInterfacesSmmServices(std::vector<json> protocols) {
    if (!init_hexrays_plugin()) {
        return false;
    }

    // Descriptors for _EFI_SMM_SYSTEM_TABLE2 functions
    struct TargetFunctionPointer SmmServicesFunctions[2]{
        {"SmmHandleProtocol", 0xb8, 3, 1, 2},
        {"SmmLocateProtocol", 0xd0, 3, 0, 2},
    };

    // Initialize
    ServiceDescriptor sdSmm;
    sdSmm.Initialize("_EFI_SMM_SYSTEM_TABLE2", SmmServicesFunctions, 2);

    ServiceDescriptorMap mSmm;
    mSmm.Register(sdSmm);

    GUIDRetyper retyperSmm(mSmm);
    retyperSmm.SetProtocols(protocols);

    // Handle all protocols
    for (auto protocol : protocols) {
        auto code_addr = protocol["ea"];
        auto service = protocol["service"];

        func_t *f = get_func(code_addr);
        if (f == nullptr) {
            continue;
        }

        retyperSmm.SetCodeEa(code_addr);
        retyperSmm.SetFuncEa(f->start_ea);

        hexrays_failure_t hf;
        cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);

        // Check that the function is decompiled
        if (cfunc == nullptr) {
            continue;
        }

        retyperSmm.apply_to(&cfunc->body, nullptr);
```

```
    }

    return true;
}

uint8_t VariablesInfoExtractAll(func_t *f, ea_t code_addr) {
    if (!init_hexrays_plugin()) {
        return 0xff;
    }

    // check func
    if (f == nullptr) {
        return 0xff;
    }
    VariablesInfoExtractor extractor(code_addr);
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    // Check that the function is decompiled
    if (cfunc == nullptr) {
        return 0xff;
    }
    extractor.apply_to(&cfunc->body, nullptr);
    auto res = extractor.mAttributes;
    return res;
}

bool TrackEntryParams(func_t *f, uint8_t depth) {
    if (!init_hexrays_plugin()) {
        return false;
    }

    if (depth == 2) {
        return true;
    }
    // check func
    if (f == nullptr) {
        return false;
    }
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    if (cfunc == nullptr) {
        return false;
    }
    PrototypesFixer *pf = new PrototypesFixer();
    pf->apply_to(&cfunc->body, nullptr);
    for (auto addr : pf->child_functions) {
        TrackEntryParams(get_func(addr), ++depth);
    }
    delete pf;

    return true;
}

json DetectVars(func_t *f) {
    json res;

    if (!init_hexrays_plugin()) {
        return res;
    }

    // check func
    if (f == nullptr) {
        return res;
    }
```

```
    VariablesDetector vars_detector;
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    if (cfunc == nullptr) {
        return res;
    }

    vars_detector.SetFuncEa(f->start_ea);
    vars_detector.apply_to(&cfunc->body, nullptr);

    res["gImageHandleList"] = vars_detector.gImageHandleList;
    res["gStList"] = vars_detector.gStList;
    res["gBsList"] = vars_detector.gBsList;
    res["gRtList"] = vars_detector.gRtList;

    return res;
}

std::vector<json> DetectServices(func_t *f) {
    // check func
    std::vector<json> res;

    if (!init_hexrays_plugin()) {
        return res;
    }

    if (f == nullptr) {
        return res;
    }
    ServicesDetector services_detector;
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    if (cfunc == nullptr) {
        return res;
    }
    services_detector.apply_to(&cfunc->body, nullptr);
    return services_detector.services;
}

bool DetectPeiServices(func_t *f) {
    if (!init_hexrays_plugin()) {
        return false;
    }

    if (f == nullptr) {
        return false;
    }

    PeiServicesDetector pei_services_detector;
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    if (cfunc == nullptr) {
        return false;
    }
    pei_services_detector.apply_to(&cfunc->body, nullptr);

    return true;
}

std::vector<json> DetectPeiServicesArm(func_t *f) {
    std::vector<json> res;

    if (!init_hexrays_plugin()) {
        return res;
```

```
    }

    if (f == nullptr) {
        return res;
    }

    PeiServicesDetectorArm pei_services_detector_arm;
    hexrays_failure_t hf;
    cfuncptr_t cfunc = decompile(f, &hf, DECOMP_NO_WAIT);
    if (cfunc == nullptr) {
        return res;
    }
    pei_services_detector_arm.apply_to(&cfunc->body, nullptr);
    return pei_services_detector_arm.services;
}
}
```

efiXplorer/efiHexRays.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly, Rolf Rolles
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiHexRays.h
 *
 */

#pragma once

#include "efiUtils.h"

uint8_t VariablesInfoExtractAll(func_t *f, ea_t code_addr);
bool TrackEntryParams(func_t *f, uint8_t depth);
json DetectVars(func_t *f);
std::vector<json> DetectServices(func_t *f);
std::vector<json> DetectPeiServicesArm(func_t *f);
bool DetectPeiServices(func_t *f);
bool setLvarName(qstring name, lvar_t lvar, ea_t func_addr);
bool applyAllTypesForInterfacesBootServices(std::vector<json> guids);
bool applyAllTypesForInterfacesSmmServices(std::vector<json> guids); // unused
bool setHexRaysVariableInfo(ea_t funcEa, lvar_t &ll, tinfo_t tif, std::string name);
bool setHexRaysVariableInfoAndHandleInterfaces(ea_t funcEa, lvar_t &ll, tinfo_t tif,
                                               std::string name);
bool offsetOf(tinfo_t tif, const char *name, unsigned int *offset);
bool isPODArray(tinfo_t tif, unsigned int ptrDepth);
const char *Expr2String(cexpr_t *e, qstring *out);
```

```
// Description of a function pointer within a structure. Ultimately, this
// plugin is looking for calls to specific UEFI functions. This structure
// describes basic information about those functions:
struct TargetFunctionPointer {
    const char *name;      // Name of function pointer in structure
    int offset;            // Offset of function pointer (filled in later)
    unsigned int nArgs;    // Number of expected arguments
    unsigned int nGUIDArg; // Which argument has the EFI_GUID *
    unsigned int nOutArg;  // Which argument retrieves the output
};

// This class holds all function pointer descriptors for one structure, as well
// as providing a utility to look up function pointers by offset.
class ServiceDescriptor {
    // Instance data
  protected:
    // The type of the containing structure (e.g. EFI_BOOT_SERVICES)
    tinfo_t mType;

    // The name of the type (e.g. "EFI_BOOT_SERVICES")
    qstring mName;

    // The ordinal of the type (e.g. 4)
    uint32 mOrdinal;

    // A vector of the structures above, copied, and with the offsets filled in
    std::vector<TargetFunctionPointer> mTargets;

    bool bInitialized;

    // Ensure we can look up the type that this instance describes
    bool InitType(const char *name) {
        // Import type
        import_type(get_idati(), -1, name);

        // Get type by name
        if (!mType.get_named_type(get_idati(), name, BTF_STRUCT))
            return false;

        // Save ordinal and name
        mOrdinal = mType.get_ordinal();
        mName = name;
        return true;
    }

    // Look up the offsets for all function pointer targets; save the results
    // in the vector. Return false if offset lookup fails.
    bool InitTargets(TargetFunctionPointer *targets, size_t num) {
        // Iterate through all targets
        for (int i = 0; i < num; ++i) {

            // Copy the target structure into our local vector
            TargetFunctionPointer &tgt = mTargets.emplace_back();
            tgt = targets[i];

            // Retrieve the offsets of each named function pointer
            unsigned int offset;
            if (!offsetOf(mType, targets[i].name, &offset)) {
                return false;
            }
        }
        return true;
    }
```

```
  public:
    // Constructor does nothing
    ServiceDescriptor() : mOrdinal(0), bInitialized(false){};

    // Accessor for ordinal
    uint32 GetOrdinal() { return mOrdinal; };

    // Accessor for name
    const char *GetName() { return mName.c_str(); };

    // Needs to be called before the object can be used
    bool Initialize(const char *name, TargetFunctionPointer *targets, size_t num) {
        if (bInitialized)
            return true;
        bInitialized = InitType(name) && InitTargets(targets, num);
        return bInitialized;
    }

    // After initialization, look up a target by offset
    bool LookupOffset(unsigned int offset, TargetFunctionPointer **tgt) {
        // Iterating through a vector generally is inefficient compared to a map,
        // but there are at most 3 function pointers so far, so it outweighs the
        // overhead of the associative containers.
        for (auto &it : mTargets) {
            // Match by offset
            if (it.offset == offset) {
                *tgt = &it;
                return true;
            }
        }
        // If we don't find it, it's not necessarily "bad" from the point of view
        // of the plugin's logic. After all, we're looking at every access to the
        // selected structures, and so, quite rightly, we'll want to ignore the
        // function pointers that we're not tracking.
        return false;
    }
};

// This class manages multiple instances of the class above. Each such
// structure is associated with the ordinal of its containing structure type.
// Then, when the Hex-Rays visitor needs to look up a function pointer access
// into a structure, it just passes the structure ordinal and offset. This
// class looks up the ServiceDescriptor in a map by ordinal, and then looks up
// the offset if that succeeded.
class ServiceDescriptorMap {
  protected:
    // Our map for looking up ServiceDescriptor structures. I should probably
    // change the value type to a pointer.
    std::map<uint32, ServiceDescriptor> mServices;

  public:
    // Add a new ServiceDescriptor to the map. I should change the argument
    // type to match whatever I change the value type of the map to.
    bool Register(ServiceDescriptor sd) {

        // Get the ordinal from the ServiceDescriptor
        uint32 ord = sd.GetOrdinal();

        // Are we already tracking this structure?
        if (mServices.find(ord) != mServices.end()) {
            return false;
        }
        // If not, register it. Get rid of std::move
        mServices[ord] = std::move(sd);
```

```
        return true;
    }

    // This function could be protected, but whatever. Given an ordinal, get
    // the tracked ServiceDescriptor, if applicable.
    bool LookupOrdinal(uint32 ord, ServiceDescriptor **sd) {
        auto it = mServices.find(ord);
        if (it == mServices.end()) {
            return false;
        }
        *sd = &it->second;
        return true;
    }

    // This is the high-level function that clients call. Given a structure
    // ordinal and offset of a function pointer, see if it's something we're
    // tracking. If so, get pointers to the tracked objects and return true.
    bool LookupOffset(uint32 ord, unsigned int offset, ServiceDescriptor **sd,
                      TargetFunctionPointer **tgt) {
        if (!LookupOrdinal(ord, sd))
            return false;
        if (!(*sd)->LookupOffset(offset, tgt))
            return false;
        return true;
    }
};

// Base class for two visitors that require similar functionality. Here we
// collect all of the common data and functionality that will be used by both
// of those visitors. This allows the derivatives to be very succinct.
class GUIDRelatedVisitorBase : public ctree_visitor_t {
  public:
    // We need access to a ServiceDescriptorMap from above.
    GUIDRelatedVisitorBase(ServiceDescriptorMap &m)
        : ctree_visitor_t(CV_FAST), mDebug(true), mServices(m){};

    // We need the function ea when setting Hex-Rays variable types.
    void SetFuncEa(ea_t ea) { mFuncEa = ea; };
    void SetCodeEa(ea_t ea) { mCodeEa = ea; };
    void SetProtocols(std::vector<json> protocols) { mProtocols = protocols; };

  protected:
    //
    // Persistent variables
    //

    // Function address
    ea_t mFuncEa;
    ea_t mCodeEa;

    // Protocols
    std::vector<json> mProtocols;

    // Print debug messages?
    bool mDebug = false;

    // Used for looking up calls to function pointers in structures
    ServiceDescriptorMap &mServices;

    //
    // State variables, cleared on every iteration. I debated with myself
    // whether this was a nasty design decision. I think it's fine. These
    // variables are only valid to access after the client has called
    // ValidateCallAndGUID, and it returned true. If you called that and it
```

```
// returned false, these will be in an inconsistent state. Don't touch them
// if that's the case.
//

// Address of the indirect function call
ea_t mEa;

// The pointer type that's being accessed (that of the structure)
tinfo_t mTif;

// The structure type, with the pointer indirection removed
tinfo_t mTifNoPtr;

// The ServiceDescriptor for the containing structure
ServiceDescriptor *mpService;

// The ordinal of the structure type
uint32 mOrdinal;

// The offset of the function pointer in the structure
unsigned int mOffset;

// Details about the target of the indirect call (e.g. name)
TargetFunctionPointer *mpTarget;

// The list of arguments for the indirect call
carglist_t *mArgs;

// The argument that specifies the GUID for the indirect call
cexpr_t *mGUIDArg;

// The argument that gets the output for the indirect call
cexpr_t *mOutArg;

// The GUID argument will be &x; this is x
cexpr_t *mGUIDArgRefTo;

// The address of the GUID being passed to the indirect call
ea_t mGUIDEa;

// This function clears all the state variables above. Technically, it
// doesn't need to exist, since the flow of logic in the functions below
// always write to them before reading to them. But, it seems like good
// programming practice not to have stale values, anyway.
void Clear() {
    mEa = BADADDR;
    mTif.clear();
    mTifNoPtr.clear();
    mpService = nullptr;
    mOrdinal = 0;
    mOffset = -1;
    mpTarget = nullptr;
    mArgs = nullptr;
    mGUIDArg = nullptr;
    mOutArg = nullptr;
    mGUIDArgRefTo = nullptr;
    mGUIDEa = BADADDR;
};

// Debug print, if the instance debug variable says to
void DebugPrint(const char *fmt, ...) {
    va_list va;
    va_start(va, fmt);
    if (mDebug)
```

```
        vmsg(fmt, va);
};

// This is the first function called every time the visitor visits an
// expression. This function determines if the expression is a call to a
// function pointer contained in a structure.
bool GetICallOrdAndOffset(cexpr_t *e) {
    // Set instance variable for call address
    mEa = e->ea;

    if (mEa != mCodeEa) {
        return false;
    }

    // If it's not a call, we're done.
    if (e->op != cot_call)
        return false;

    // Set instance variable with call arguments
    mArgs = e->a;

    // If it's a direct call, we're done.
    cexpr_t *callDest = e->x;
    if (callDest->op == cot_obj)
        return false;

    // Eat any casts on the type of what's being called
    while (callDest->op == cot_cast)
        callDest = callDest->x;

    // If the destination is not a member of a structure, we're done.
    if (callDest->op != cot_memptr)
        return false;

    // Set instance variable with type of structure containing pointer
    mTif = callDest->x->type;

    // Ensure that the structure is being accessed via pointer, and not as a
    // reference (i.e., through a structure held on the stack as a local
    // variable).
    if (!mTif.is_ptr()) {
        return false;
    }

    // Remove pointer from containing structure type, set instance variable
    mTifNoPtr = remove_pointer(mTif);

    // Get the ordinal of the structure
    mOrdinal = mTifNoPtr.get_ordinal();

    // If we can't get a type for the structure, that's bad
    if (mOrdinal == 0)
        return false;

    // Get the offset of the function pointer in the structure
    mOffset = callDest->m;

    // Okay: now we know we're dealing with an indirect call to a function
    // pointer contained in a structure, where the structure is being
    // accessed by a pointer.
    return true;
};

// This is the second function called as part of indirect call validation.
```

```
// Now we want to know: is it a call to something that we're tracking?
bool ValidateICallDestination() {

    // Look up the structure ordinal and function offset; get the associated
    // ServiceDescriptor and TargetFunctionPointer (instance variables).
    if (!mServices.LookupOffset(mOrdinal, mOffset, &mpService, &mpTarget))
        return false;

    // Great, it was something that we were tracking. Now, sanity-check the
    // number of arguments on the function call. (Hex-Rays might have gotten
    // this wrong. The user can fix it via "set call type".)
    size_t mArgsSize = mArgs->size();
    size_t nArgs = mpTarget->nArgs;
    if (mArgsSize != nArgs) {
        return false;
    }

    // The TargetFunctionPointer tells us which argument takes an EFI_GUID *,
    // and which one retrieves the output. Get those arguments, and save them
    // as instance variables.
    mGUIDArg = &mArgs->at(mpTarget->nGUIDArg);
    mOutArg = &mArgs->at(mpTarget->nOutArg);

    // Great; now we know that the expression is an indirect call to
    // something that we're tracking, and that Hex-Rays decompiled the call
    // the way we expected it to.
    return true;
};

// This is a helper function used to get the thing being referred to. What
// does that mean?
//
// * For GUID arguments, we'll usually have &globvar. Return globvar.
// * For output arguments, we'll usually have &globvar or &locvar. Due to
//   Hex-Rays internal heuristics, we might end up with "locarray", which
//   does not actually have a "&" when passed as a call argument. There's
//   a bit of extra logic to check for that case.
cexpr_t *GetReferent(cexpr_t *e, const char *desc, bool bAcceptVar) {

    // Eat casts
    cexpr_t *x = e;
    while (x->op == cot_cast)
        x = x->x;

    qstring estr;
    // If we're accepting local variables, and this is a variable (note: not
    // a *reference* to a variable)
    if (bAcceptVar && x->op == cot_var) {
        // Get the variable details
        var_ref_t varRef = x->v;
        lvar_t destVar = varRef.mba->vars[varRef.idx];

        // Ensure that it's an array of POD types, or pointers to them
        bool bisPODArray = isPODArray(destVar.tif, 1);

        // If it is a POD array, good, we'll take it.
        return bisPODArray ? x : nullptr;
    }

    // For everything else, we really want it to be a reference: either to a
    // global or local variable. If it's not a reference, we can't get the
    // referent, so fail.
    if (x->op != cot_ref) {
        return nullptr;
```

```
        }

        // If we get here, we know it's a reference. Return the referent.
        return x->x;
    };

    // The third function in the validation logic. We already know the
    // expression is an indirect call to something that we're tracking, and
    // that Hex-Rays' decompilation matches on the number of arguments. Now,
    // we validate that the GUID argument does in fact point to a global
    // variable.
    bool ValidateGUIDArgument() {
        // Does the GUID argument point to a local variable?
        mGUIDArgRefTo = GetReferent(mGUIDArg, "GUID", false);
        if (!mGUIDArgRefTo)
            return false;

        // If we get here, we know it was a reference to *something*. Ensure that
        // something is a global variable.
        if (mGUIDArgRefTo->op != cot_obj) {
            return false;
        }

        // Save the address of the global variable to which the GUID argument is
        // pointing.
        mGUIDEa = mGUIDArgRefTo->obj_ea;

        // Great; now we know we're dealing with an indirect call to something
        // we're tracking; that Hex-Rays decompiled the call with the proper
        // number of arguments; and that the GUID argument did in fact point to
        // a global variable, whose address we now have in an instance variable.
        return true;
    };

    // Finally, this function combines all three checks above into one single
    // function. If you call this and it returns true, feel free to access the
    // instance variables, as they are guaranteed to be valid. If it returns
    // false, they aren't, so don't touch them.
    bool ValidateCallAndGUID(cexpr_t *e) {
        // Reset all instance variables. Not strictly necessary; call it
        // "defensive programming".
        Clear();

        // Validate according to the logic above.
        if (!GetICallOrdAndOffset(e) || !ValidateICallDestination() ||
            !ValidateGUIDArgument())
            return false;

        // Good, all checks passed
        return true;
    }
};

// Now that we've implemented all that validation logic, this class is pretty
// simple. This one is responsible for ensuring that the GUID is something that
// we know about, and setting the types of the output variables accordingly.
class GUIDRetyper : public GUIDRelatedVisitorBase {
  public:
    GUIDRetyper(ServiceDescriptorMap &m) : GUIDRelatedVisitorBase(m), mNumApplied(0){};

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        // Perform the checks from GUIDRelatedVisitorBase. If they fail, we're
```

```
        // not equipped to deal with this expression, so bail out.
        if (!ValidateCallAndGUID(e))
            return 0;

        mGUIDArgRefTo = GetReferent(mGUIDArg, "GUID", false);
        if (mGUIDArgRefTo == nullptr)
            return 0;
        ea_t guidAddr = mGUIDArgRefTo->obj_ea;

        // Get interface type name
        std::string GUIDName;
        for (auto g : mProtocols) {
            if (guidAddr == g["address"]) {
                GUIDName = g["prot_name"];
                break;
            }
        }
        if (GUIDName.empty()) {
            return 0;
        }

        std::string interfaceTypeName = GUIDName.substr(0, GUIDName.find("_GUID"));
        if (!interfaceTypeName.find("FCH_")) {
            // convert FCH_SMM_* dispatcher type to EFI_SMM_* dispatcher type
            interfaceTypeName.replace(0, 4, "EFI_");
        }

        // Need to get the type for the interface variable here
        tinfo_t tif;
        import_type(get_idati(), -1, interfaceTypeName.c_str());
        if (!tif.get_named_type(get_idati(), interfaceTypeName.c_str())) {
            // Get the referent for the interface argument.
            cexpr_t *outArgReferent = GetReferent(mOutArg, "ptr", true);
            if (outArgReferent == nullptr)
                return 0;
            ApplyName(outArgReferent, interfaceTypeName);
            return 0;
        }

        qstring tStr;
        if (!tif.get_type_name(&tStr)) {
            return 0;
        }

        tinfo_t tifGuidPtr;
        if (!tifGuidPtr.create_ptr(tif)) {
            return 0;
        }

        // Get the referent for the interface argument.
        cexpr_t *outArgReferent = GetReferent(mOutArg, "ptr", true);
        if (outArgReferent == nullptr)
            return 0;

        // Apply the type to the output referent.
        ApplyType(outArgReferent, tifGuidPtr, tStr);
        return 1;
    }

protected:
    unsigned int mNumApplied;

    // Given an expression (either a local or global variable) and a type to
    // apply, apply the type. This is just a bit of IDA/Hex-Rays type system
```

```
    // skullduggery.
    void ApplyType(cexpr_t *outArg, tinfo_t ptrTif, qstring tStr) {
        ea_t dest_ea = outArg->obj_ea;

        // For global variables
        if (outArg->op == cot_obj) {
            // Just apply the type information to the address
            apply_tinfo(dest_ea, ptrTif, TINFO_DEFINITE);
            ++mNumApplied;

            // Rename global variable
            auto name = "g" + typeToName(static_cast<std::string>(tStr.c_str()));
            set_name(dest_ea, name.c_str(), SN_FORCE);

            // Get xrefs to global variable
            auto xrefs = getXrefs(dest_ea);
            qstring typeName;
            ptr_type_data_t pi;
            ptrTif.get_ptr_details(&pi);
            pi.obj_type.get_type_name(&typeName);
            // Handling all interface functions (to rename function arguments)
            opstroffForGlobalInterface(xrefs, typeName);
        }

        // For local variables
        else if (outArg->op == cot_var) {
            var_ref_t varRef = outArg->v;
            lvar_t &destVar = varRef.mba->vars[varRef.idx];
            // Set the Hex-Rays variable type
            auto name = typeToName(static_cast<std::string>(tStr.c_str()));
            setLvarName(static_cast<qstring>(name.c_str()), destVar, mFuncEa);
            if (setHexRaysVariableInfoAndHandleInterfaces(mFuncEa, destVar, ptrTif,
                                                          name)) {
                ++mNumApplied;
            }
        }
    }

    void ApplyName(cexpr_t *outArg, std::string type_name) {
        ea_t dest_ea = outArg->obj_ea;

        // For global variables
        if (outArg->op == cot_obj) {
            // Rename global variable
            auto name = "g" + typeToName(type_name);
            set_name(dest_ea, name.c_str(), SN_FORCE);
        }

        // For local variables
        else if (outArg->op == cot_var) {
            var_ref_t varRef = outArg->v;
            lvar_t &destVar = varRef.mba->vars[varRef.idx];
            // Set the Hex-Rays variable type
            auto name = typeToName(type_name);
            setLvarName(static_cast<qstring>(name.c_str()), destVar, mFuncEa);
        }
    }
};

class VariablesInfoExtractor : public ctree_visitor_t {
  public:
    VariablesInfoExtractor(ea_t code_addr) : ctree_visitor_t(CV_FAST) {
        mCodeAddr = code_addr;
    };
```

```
    uint8_t mAttributes = 0xff;

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        if (mCodeAddr == BADADDR) {
            return 0;
        }

        if (e->ea != mCodeAddr) {
            return 0;
        }

        if (e->op != cot_call)
            return 0;

        carglist_t *args = e->a;
        if (args == nullptr) {
            return 0;
        }

        size_t args_size = args->size();
        if (args_size < 3) {
            return 0;
        }

        cexpr_t *attributes_arg = &args->at(2);
        if (attributes_arg->op == cot_num) {
            if (mDebug) {
                msg("[I] Service call: %016llX, Attributes:
%02X\n", u64_addr(mCodeAddr),
                    static_cast<uint8_t>(attributes_arg->numval()));
            }
            attributes_arg->numval();
            mAttributes = static_cast<uint8_t>(attributes_arg->numval());
        }

        return 0;
    }

  protected:
    ea_t mCodeAddr = BADADDR;
    bool mDebug = false;
};

class PrototypesFixer : public ctree_visitor_t {
  public:
    PrototypesFixer() : ctree_visitor_t(CV_FAST){};
    std::vector<ea_t> child_functions;

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        if (e->op != cot_call)
            return 0;

        // get child function address
        if (e->x->op != cot_obj) {
            return 0;
        }
        if (mDebug) {
            msg("[I] Child function address: %016llX\n", u64_addr(e->x->obj_ea));
        }
```

```
        carglist_t *args = e->a;
        if (args == nullptr) {
            return 0;
        }

        // get child function prototype
        ea_t func_addr = e->x->obj_ea;
        hexrays_failure_t hf;
        func_t *f = get_func(func_addr);
        if (f == nullptr) {
            return 0;
        }

        cfuncptr_t cf = decompile(f, &hf, DECOMP_NO_WAIT);
        if (cf == nullptr) {
            return 0;
        }

        msg("[I] Call address: 0x%016llX\n", u64_addr(e->ea));
        for (auto i = 0; i < args->size(); i++) {
            cexpr_t *arg = &args->at(i);
            if (arg->op == cot_cast || arg->op == cot_var) {
                // extract argument type
                tinfo_t arg_type;
                tinfo_t arg_type_no_ptr;
                if (arg->op == cot_var) {
                    arg_type = arg->type;
                }
                if (arg->op == cot_cast) {
                    arg_type = arg->x->type;
                }

                // print type
                if (arg_type.is_ptr()) {
                    arg_type_no_ptr = remove_pointer(arg_type);
                }

                qstring type_name;
                bool is_ptr = false;
                if (!arg_type.get_type_name(&type_name)) {
                    if (!arg_type_no_ptr.get_type_name(&type_name)) {
                        // msg("[E] Can not get type name: 0x%016llX\n",
u64_addr(e->ea));
                        continue;
                    }
                    is_ptr = true;
                }

                if (is_ptr) {
                    msg("[I]  Arg #%d, type =  %s *\n", i, type_name.c_str());
                } else {
                    msg("[I]  Arg #%d, type =  %s\n", i, type_name.c_str());
                }

                if (type_name == qstring("EFI_HANDLE") ||
                    type_name == qstring("EFI_SYSTEM_TABLE")) {
                    if (!addrInVec(child_functions, func_addr)) {
                        child_functions.push_back(func_addr);
                    }
                    // set argument type and name
                    if (cf->argidx.size() <= i) {
                        return 0;
                    }
```

```
                    auto argid = cf->argidx[i];
                    lvar_t &arg_var = cf->mba->vars[argid]; // get lvar for argument
                    if (type_name == qstring("EFI_HANDLE")) {
                        setHexRaysVariableInfo(func_addr, arg_var, arg_type,
                                               "ImageHandle");
                    }
                    if (type_name == qstring("EFI_SYSTEM_TABLE")) {
                        setHexRaysVariableInfo(func_addr, arg_var, arg_type,
                                               "SystemTable");
                    }
                }
            }
        }

        return 0;
    }

  protected:
    bool mDebug = true;
};

class VariablesDetector : public ctree_visitor_t {
  public:
    VariablesDetector() : ctree_visitor_t(CV_FAST){};

    std::vector<ea_t> child_functions;

    std::vector<ea_t> gImageHandleList;
    std::vector<ea_t> gStList;
    std::vector<ea_t> gBsList;
    std::vector<ea_t> gRtList;

    void SetFuncEa(ea_t ea) { mFuncEa = ea; };

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        if (e->op == cot_asg) {
            // saving a child function for recursive analysis
            if (!addrInVec(child_functions, e->ea)) {
                child_functions.push_back(e->x->obj_ea);
            }
        }

        bool global_var = false;
        bool local_var = false;
        if (e->op != cot_asg) {
            return 0;
        }

        switch (e->x->op) {
        case cot_obj:
            // asg operation for global variable
            global_var = true;
            break;
        case cot_var:
            // asg operation for local variable
            local_var = true;
            break;
        default:
            return 0;
        }

        if (e->y->op != cot_cast && e->y->op != cot_var) {
```

```
    return 0;
}

// extract variable type
tinfo_t var_type;
tinfo_t var_type_no_ptr;
if (e->y->op == cot_var) {
    var_type = e->y->type;
}
if (e->y->op == cot_cast) {
    var_type = e->y->x->type;
}

if (var_type.is_ptr()) {
    var_type_no_ptr = remove_pointer(var_type);
}

qstring type_name;
bool is_ptr = false;
if (!var_type.get_type_name(&type_name)) {
    if (!var_type_no_ptr.get_type_name(&type_name)) {
        // msg("[E] can not get type name: 0x%016llX\n", u64_addr(e->ea));
        return 0;
    }
    is_ptr = true;
}

if (mDebug) {
    msg("[I] code address: 0x%016llX, type name: %s\n", u64_addr(e->ea),
        type_name.c_str());
}

if (global_var) {
    // extract variable data
    ea_t g_addr = e->x->obj_ea;
    std::string type_name_str = static_cast<std::string>(type_name.c_str());
    if (type_name == qstring("EFI_HANDLE")) {
        setTypeAndName(g_addr, "gImageHandle", type_name_str);
        if (!addrInVec(gImageHandleList, g_addr)) {
            gImageHandleList.push_back(g_addr);
        }
    }
    if (type_name == qstring("EFI_SYSTEM_TABLE")) {
        setPtrTypeAndName(g_addr, "gST", type_name_str);
        if (!addrInVec(gStList, g_addr)) {
            gStList.push_back(g_addr);
        }
    }
    if (type_name == qstring("EFI_BOOT_SERVICES")) {
        setPtrTypeAndName(g_addr, "gBS", type_name_str);
        if (!addrInVec(gBsList, g_addr)) {
            gBsList.push_back(g_addr);
        }
    }
    if (type_name == qstring("EFI_RUNTIME_SERVICES")) {
        setPtrTypeAndName(g_addr, "gRT", type_name_str);
        if (!addrInVec(gRtList, g_addr)) {
            gRtList.push_back(g_addr);
        }
    }
}

if (local_var) {
    var_ref_t var_ref;
```

```
            if (e->y->op == cot_var) {
                var_ref = e->y->v;
            }
            if (e->y->op == cot_cast) {
                var_ref = e->y->x->v;
            }
            lvar_t &dest_var = var_ref.mba->vars[var_ref.idx];
            // Set the Hex-Rays variable type
            auto name = typeToName(static_cast<std::string>(type_name.c_str()));
            // setHexRaysVariableInfo(mFuncEa, dest_var, var_type, name);
        }

        return 0;
    }

  protected:
    bool mDebug = true;
    ea_t mFuncEa = BADADDR;
};

class ServicesDetector : public ctree_visitor_t {
    // detect all services (Boot services, Runtime services, etc)
  public:
    ServicesDetector() : ctree_visitor_t(CV_FAST){};

    std::vector<json> services;

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        if (e->op != cot_call) {
            return 0;
        }

        if (e->x->op != cot_cast) {
            return 0;
        }

        // extract function type
        auto e_func = e->x->x;
        tinfo_t func_type;
        tinfo_t func_type_no_ptr;
        func_type = e_func->type;

        if (func_type.is_ptr()) {
            func_type_no_ptr = remove_pointer(func_type);
        }

        qstring type_name;
        bool is_ptr = false;
        if (!func_type.get_type_name(&type_name)) {
            if (!func_type_no_ptr.get_type_name(&type_name)) {
                // msg("[E] can not get type name: 0x%016llX\n", u64_addr(e->ea));
                return 0;
            }
            is_ptr = 0;
        }

        auto service_name = typeToName(static_cast<std::string>(type_name.c_str()));
        if (service_name.rfind("Efi", 0) == 0) {
            service_name = service_name.substr(3);
            if (service_name == "RaiseTpl") {
                service_name = "RaiseTPL";
            }
```

```
            if (service_name == "RestoreTpl") {
                service_name = "RestoreTPL";
            }
        }
        msg("[efiXplorer] address: 0x%016llX, service type: %s, service name: %s\n",
            u64_addr(e->ea), type_name.c_str(), service_name.c_str());

        // append service
        // add item to allBootServices
        json s;
        s["address"] = e->ea;
        s["service_name"] = service_name;
        s["table_name"] = getTable(service_name);

        if (!jsonInVec(services, s)) {
            services.push_back(s);
        }

        return 0;
    }

  protected:
    bool mDebug = true;
};

class PeiServicesDetector : public ctree_visitor_t {
    // detect and mark all PEI services
  public:
    PeiServicesDetector() : ctree_visitor_t(CV_FAST){};

    bool make_shifted_ptr(tinfo_t outer, tinfo_t inner, int32 offset,
                          tinfo_t *shifted_tif) {
        ptr_type_data_t pi;
        pi.taptr_bits = TAPTR_SHIFTED;
        pi.delta = offset;
        pi.parent = outer;
        pi.obj_type = inner;
        shifted_tif->create_ptr(pi);
        return shifted_tif->is_correct();
    }

    bool set_var_type(ea_t func_ea, lvar_t lvar, tinfo_t tif) {
        lvar_saved_info_t lsi;
        lsi.ll = lvar;
        lsi.type = tif;
        return modify_user_lvar_info(func_ea, MLI_TYPE, lsi);
    }

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
    int visit_expr(cexpr_t *e) {
        ea_t pointer_offset = BADADDR;
        ea_t service_offset = BADADDR;
        bool call = false;
        var_ref_t var_ref;
        if (e->op == cot_ptr && e->x->op == cot_cast && e->x->x->op == cot_add &&
            e->x->x->x->op == cot_ptr && e->x->x->x->x->op == cot_ptr &&
            e->x->x->x->x->x->x->op == cot_cast && e->x->x->x->x->x->x->x->op == cot_sub &&
            e->x->x->x->x->x->x->x->x->op == cot_var &&
            e->x->x->x->x->x->x->x->y->op == cot_num && e->x->x->y->op == cot_num) {
            // (*ADJ(v2)->PeiServices)->GetHobList((const EFI_PEI_SERVICES
            // **)ADJ(v2)->PeiServices, HobList);
            service_offset = e->x->x->y->numval();
            pointer_offset = e->x->x->x->x->x->x->y->numval();
```

```
            var_ref = e->x->x->x->x->x->x->v;
            call = true;
        } else if (e->op == cot_asg && e->x->op == cot_var && e->y->op == cot_ptr &&
                   e->y->x->op == cot_cast && e->y->x->x->op == cot_sub &&
                   e->y->x->x->x->op == cot_var && e->y->x->x->y->op == cot_num) {
            // __sidt(v6);
            // PeiServices = ADJ(v7)->PeiServices;
            pointer_offset = e->y->x->x->y->numval();
            var_ref = e->y->x->x->x->v;
        } else {
            return 0;
        }

        msg("[efiXplorer] address: 0x%08llX, PEI service detected\n", u64_addr(e->ea));
        msg("[efiXplorer]   delta: %llx\n", u64_addr(pointer_offset));
        if (service_offset != BADADDR) {
            msg("[efiXplorer]   service offset: %llx\n", u64_addr(service_offset));
        }

        if (pointer_offset != 4) {
            // handle only 4 for now
            return 0;
        }

        tinfo_t outer;
        if (!outer.get_named_type(get_idati(), "EFI_PEI_SERVICES_4", BTF_STRUCT)) {
            return 0;
        }

        tinfo_t shifted_tif;
        if (!make_shifted_ptr(outer, outer, pointer_offset, &shifted_tif)) {
            return 0;
        }

        lvar_t &dest_var = var_ref.mba->vars[var_ref.idx];
        func_t *func = get_func(e->ea);
        if (func == nullptr) {
            return 0;
        }
        if (set_var_type(func->start_ea, dest_var, shifted_tif)) {
            msg("[efiXplorer] shifted pointer applied (0x%08llX)\n", u64_addr(e->ea));
        }

        if (call) {
            opStroff(e->ea, "EFI_PEI_SERVICES");
        }

        return 0;
    }

  protected:
    bool mDebug = true;
};

class PeiServicesDetectorArm : public ctree_visitor_t {
    // detect and mark all PEI services for ARM firmware
    // tested on Ampere firmware that contains small PEI stack
  public:
    PeiServicesDetectorArm() : ctree_visitor_t(CV_FAST){};

    std::vector<json> services;

    // This is the callback function that Hex-Rays invokes for every expression
    // in the CTREE.
```

```
  int visit_expr(cexpr_t *e) {
      if (!(e->op == cot_call && e->x->op == cot_memptr && e->x->x->op == cot_ptr &&
            e->x->x->x->op == cot_var)) {
          return 0;
      }
      ea_t offset = e->x->m;

      // check if service from EFI_PEI_SERVICES
      tinfo_t table_type = e->x->x->type;
      tinfo_t table_type_no_ptr;
      qstring table_type_name;
      if (table_type.is_ptr()) {
          table_type_no_ptr = remove_pointer(table_type);
          table_type_no_ptr.get_type_name(&table_type_name);
      } else {
          table_type.get_type_name(&table_type_name);
      }

      // get service name from function type
      std::string service_name;
      if (table_type_name != "EFI_PEI_SERVICES") {
          qstring func_type_name;
          tinfo_t service_type = e->x->type;
          service_type.get_type_name(&func_type_name);
          std::string func_type = static_cast<std::string>(func_type_name.c_str());
          std::string prefix = "EFI_PEI_";
          if (func_type.substr(0, prefix.length()) == prefix) {
              func_type.erase(0, prefix.length());
          }
          service_name = typeToName(func_type);
      } else {
          auto s = mPeiServices.find(offset);
          if (s == mPeiServices.end()) {
              return 0;
          }
          service_name = s->second;
      }
      if (mDebug) {
          msg("[efiXplorer] 0x%08llX: %s service detected (offset: %d): %s\n",
              u64_addr(e->ea), table_type_name.c_str(), u32_addr(offset),
              service_name.c_str());
      }

      json s;
      s["address"] = e->ea;
      s["service_name"] = service_name;
      s["table_name"] = table_type_name.c_str();

      if (!jsonInVec(services, s)) {
          services.push_back(s);
      }

      return 0;
  }

protected:
  bool mDebug = true;
  std::map<ea_t, std::string> mPeiServices = {
      {0x18, "InstallPpi"},
      {0x20, "ReInstallPpi"},
      {0x28, "LocatePpi"},
      {0x30, "NotifyPpi"},
      {0x38, "GetBootMode"},
      {0x40, "SetBootMode"},
```

```
        {0x48, "GetHobList"},
        {0x50, "CreateHob"},
        {0x58, "FfsFindNextVolume"},
        {0x60, "FfsFindNextFile"},
        {0x68, "FfsFindSectionData"},
        {0x70, "InstallPeiMemory"},
        {0x78, "AllocatePages"},
        {0x80, "AllocatePool"},
        {0x88, "CopyMem"},
        {0x90, "SetMem"},
        {0x98, "ReportStatusCode"},
        {0xA0, "ResetSystem"},
        {0xA8, "CpuIo"},
        {0xB0, "PciCfg"},
        {0xB8, "FfsFindFileByName"},
        {0xC0, "FfsGetFileInfo"},
        {0xC8, "FfsGetVolumeInfo"},
        {0xD0, "RegisterForShadow"},
        {0xD8, "FindSectionData4"},
        {0xE0, "FfsGetFileInfo3"},
        {0xE8, "ResetSystem3"},
    };
};
}
```

efiXplorer/efiSmmUtils.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiSmmUtils.cpp
 *
 */

#include "efiSmmUtils.h"

static const char plugin_name[] = "efiXplorer";

//--------------------------------------------------------------------------
// Find and mark gSmst global variable via EFI_SMM_SW_DISPATCH(2)_PROTOCOL_GUID
std::vector<ea_t> findSmstSwDispatch(std::vector<ea_t> gBsList) {
    std::vector<ea_t> smst_addrs;
    EfiGuid guid2 = {0x18a3c6dc,
                     0x5eea,
                     0x48c8,
                     {0xa1, 0xc1, 0xb5, 0x33, 0x89, 0xf9, 0x89,
```

```
                    0x99}}; // EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID
    EfiGuid guid = {0xe541b773,
                    0xdd11,
                    0x420c,
                    {0xb0, 0x26, 0xdf, 0x99, 0x36, 0x53, 0xf8,
                     0xbf}}; // EFI_SMM_SW_DISPATCH_PROTOCOL_GUID
    std::vector<ea_t> data_addrs = findData(0, BADADDR, guid.uchar_data().data(), 16);
    std::vector<ea_t> data2_addrs = findData(0, BADADDR, guid2.uchar_data().data(),
16);
    data_addrs.insert(data_addrs.end(), data2_addrs.begin(), data2_addrs.end());
    for (auto data_addr : data_addrs) {
        msg("[%s] EFI_SMM_SW_DISPATCH(2)_PROTOCOL_GUID: 0x%016llX\n", plugin_name,
            u64_addr(data_addr));
        std::vector<ea_t> xrefs = getXrefs(data_addr);
        insn_t insn;
        for (auto xref : xrefs) {
            uint16_t smst_reg = 0xffff; // Smst register
            ea_t cur_addr = xref;
            while (true) {
                cur_addr = next_head(cur_addr, BADADDR);
                decode_insn(&insn, cur_addr);
                // check for SmmLocateProtocol function call
                if (insn.itype == NN_callni && insn.ops[0].type == o_displ &&
                    insn.ops[0].addr == 0xd0) {
                    smst_reg = insn.ops[0].reg;
                    break;
                }
                if (is_basic_block_end(insn, false)) {
                    break;
                }
            }

            if (smst_reg == 0xffff) {
                continue; // smst_reg not found
            }

            ea_t res_addr = BADADDR;
            cur_addr = xref;
            while (true) {
                cur_addr = prev_head(cur_addr, 0);
                decode_insn(&insn, cur_addr);
                if (insn.itype == NN_mov && insn.ops[0].type == o_reg &&
                    insn.ops[0].reg == smst_reg && insn.ops[1].type == o_mem) {
                    msg("[%s] found gSmst at 0x%016llX, address = 0x%016llX\n",
                        plugin_name, u64_addr(cur_addr), u64_addr(insn.ops[1].addr));
                    res_addr = insn.ops[1].addr;
                    if (addrInVec(gBsList, res_addr)) {
                        continue;
                    }
                    set_cmt(cur_addr, "_EFI_SMM_SYSTEM_TABLE2 *gSmst;", true);
                    setPtrTypeAndName(res_addr, "gSmst", "_EFI_SMM_SYSTEM_TABLE2");
                    smst_addrs.push_back(res_addr);
                    break;
                }
                if (is_basic_block_end(insn, false)) {
                    break;
                }
            }
        }
    }

    return smst_addrs;
}
```

```
//------------------------------------------------------------------------
// Find and mark gSmst global variable via EFI_SMM_BASE2_PROTOCOL_GUID
std::vector<ea_t> findSmstSmmBase(std::vector<ea_t> gBsList) {
    std::vector<ea_t> smst_addrs;
    EfiGuid guid = {
        0xf4ccbfb7,
        0xf6e0,
        0x47fd,
        {0x9d, 0xd4, 0x10, 0xa8, 0xf1, 0x50, 0xc1, 0x91}};
// EFI_SMM_BASE2_PROTOCOL_GUID
    std::vector<ea_t> data_addrs = findData(0, BADADDR, guid.uchar_data().data(), 16);
    for (auto data_addr : data_addrs) {
        msg("[%s] EFI_SMM_BASE2_PROTOCOL_GUID: 0x%016llX\n", plugin_name,
            u64_addr(data_addr));
        std::vector<ea_t> data_xrefs = getXrefs(data_addr);
        insn_t insn;
        for (auto xref : data_xrefs) {
            ea_t res_addr = BADADDR;
            ea_t cur_addr = xref;
            bool in_smram = false;
            // Check 16 instructions below
            for (auto i = 0; i < 16; i++) {
                cur_addr = next_head(cur_addr, BADADDR);
                decode_insn(&insn, cur_addr);
                if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
                    insn.ops[0].reg == REG_RDX && insn.ops[1].type == o_mem) {
                    res_addr = insn.ops[1].addr;
                    msg("[%s] found gSmst/gInSmram at 0x%016llX, address
= 0x%016llX\n",
                        plugin_name, u64_addr(cur_addr), u64_addr(res_addr));
                }
                if (res_addr != BADADDR && insn.itype == NN_callni &&
                    insn.ops[0].type == o_phrase && !insn.ops[0].addr) {
                    // gEfiSmmBase2Protocol->InSmm(gEfiSmmBase2Protocol, &gInSmram)
                    in_smram = true;
                }
            }
            if (!in_smram) {
                // we found gSmst
                if (addrInVec(gBsList, res_addr)) {
                    continue;
                }
                set_cmt(cur_addr, "_EFI_SMM_SYSTEM_TABLE2 *gSmst;", true);
                setPtrTypeAndName(res_addr, "gSmst", "_EFI_SMM_SYSTEM_TABLE2");
                smst_addrs.push_back(res_addr);
            } else {
                // we found gInSmram
                setTypeAndName(res_addr, "gInSmram", "BOOLEAN");
            }
        }
    }

    return smst_addrs;
}

//------------------------------------------------------------------------
// Find SmiHandler in reg_smi_func function (prefix: Sw, TrapIo, Sx, Gpi, Usb,
// StandbyButton, PeriodicTimer, PowerButton)
std::vector<func_t *> findSmiHandlers(ea_t address, std::string prefix) {
    msg("[%s] Analyze xref to gEfiSmm%sDispatch(2)Protocol: 0x%016llX\n", plugin_name,
        prefix.c_str(), u64_addr(address));

    std::vector<func_t *> smiHandlers;
    insn_t insn;
```

```
// Find Dispatch interface address (via gSmst->SmmLocateProtocol call)

// Check instruction
decode_insn(&insn, address);
if (!(insn.ops[0].type == o_reg && insn.ops[0].reg == REG_RCX)) {
    msg("[%s] %sSmiHandler: wrong xref to dispatch(2) protocol\n", plugin_name,
        prefix.c_str());
    return smiHandlers;
}

// Analyze current basic block
auto ea = address;

// Search for SmmLocateProtocol
bool found = false;
uint64_t dispatch_interface = BADADDR;
while (!is_basic_block_end(insn, false)) {
    ea = next_head(ea, BADADDR);
    decode_insn(&insn, ea);
    if (insn.itype == NN_callni && insn.ops[0].type == o_displ &&
        insn.ops[0].addr == 0xd0) {
        found = true;
        msg("[%s] %sSmiHandler: found = true\n", plugin_name, prefix.c_str());
        break;
    }
    // Interface in stack
    if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
        insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_displ &&
        (insn.ops[1].reg == REG_RBP || insn.ops[1].reg == REG_RSP)) {
        if (dispatch_interface == BADADDR) {
            dispatch_interface = insn.ops[1].addr;
        }
    }
    // Interface in data
    if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
        insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_mem) {
        if (dispatch_interface == BADADDR) {
            dispatch_interface = insn.ops[1].addr;
        }
    }
}

if (!found) {
    return smiHandlers;
}

if (dispatch_interface == BADADDR) {
    ea = address;
    while (!is_basic_block_end(insn, false)) {
        ea = prev_head(ea, 0);
        decode_insn(&insn, ea);
        // Interface in stack
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_displ &&
            (insn.ops[1].reg == REG_RBP || insn.ops[1].reg == REG_RSP)) {
            dispatch_interface = insn.ops[1].addr;
            break;
        }
        // Interface in data
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_mem) {
            dispatch_interface = insn.ops[1].addr;
            break;
```

```
        }
    }
}

if (dispatch_interface == BADADDR) {
    return smiHandlers;
}

msg("[%s] Found EfiSmm%sDispatch(2)Protocol interface: 0x%016llX\n", plugin_name,
    prefix.c_str(), dispatch_interface);

// TODO: handle xrefs for globals
// (fw71.bin.out/SmmHddSecurity-316b1230-0500-4592-8c09-eaba0fb6b07f.smm)

// Track interface stack variable
ea = address;
uint16_t reg = BAD_REG;
uint64_t dispatch_func = BADADDR;
for (auto i = 0; i < 100; i++) {
    ea = next_head(ea, BADADDR);
    decode_insn(&insn, ea);
    // get Interface base register
    if (insn.itype == NN_mov && insn.ops[0].type == o_reg &&
        (insn.ops[1].type == o_displ || insn.ops[1].type == o_mem)) {
        if (insn.ops[1].addr == dispatch_interface) {
            reg = insn.ops[0].reg;
        } else {
            reg = BAD_REG; // resetting
        }
        continue;
    }

    // resetting (register overwrite or call)
    if (reg != BAD_REG && insn.ops[0].type == o_reg && insn.ops[0].reg == reg) {
        reg = BAD_REG;
        continue;
    }

    // resetting (call)
    if (insn.itype == NN_call) {
        reg = BAD_REG;
        continue;
    }

    // get DispatchFunction address
    if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
        insn.ops[0].reg == REG_RDX && insn.ops[1].type == o_mem) {
        dispatch_func = insn.ops[1].addr;
        continue;
    }
    if (insn.itype == NN_callni && insn.ops[0].type == o_phrase &&
        insn.ops[0].reg == reg) {
        msg("[%s] Found EfiSmm%sDispatch2Protocol->Register call (0x%016llX)\n",
            plugin_name, prefix.c_str(), u64_addr(ea));
        msg("[%s]  %sSmiHandler: 0x%016llX\n", plugin_name, prefix.c_str(),
            dispatch_func);
        auto handler_func = get_func(dispatch_func);
        if (handler_func == nullptr) {
            add_func(dispatch_func);                // create function
            handler_func = get_func(dispatch_func); // retry
        }
        if (handler_func != nullptr) {
            smiHandlers.push_back(handler_func); // add in result
        }
```

```
            reg = BAD_REG; // resetting

            // op_stroff + set_name
            std::string name = prefix + "SmiHandler";
            set_name(dispatch_func, name.c_str(), SN_FORCE);
            std::string prefix_upper;
            std::transform(prefix.begin(), prefix.end(),
prefix_upper.begin(), ::toupper);
            std::string type = "EFI_SMM_" + prefix_upper + "_DISPATCH2_PROTOCOL";
            opStroff(ea, type);
        }

        if (insn.itype == NN_retn || insn.itype == NN_int3) {
            break;
        }
    }

    return smiHandlers;
}

//----------------------------------------------------------------------
// Find {Prefix}SmiHandler function inside SMM drivers
//  * find GUID
//  * get xrefs to GUID
//  * xref will be inside RegSwSmi function
//  * find SmiHandler by pattern (instructions may be out of order)
//        lea     r9, ...
//        lea     r8, ...
//        lea     rdx, <func>
//        call    qword ptr [...]
std::vector<func_t *> findSmiHandlersSmmDispatch(EfiGuid guid, std::string prefix) {
    std::vector<func_t *> smiHandlers;
    std::vector<ea_t> data_addrs = findData(0, BADADDR, guid.uchar_data().data(), 16);
    msg("[%s] %sSmiHandler function finding\n", plugin_name, prefix.c_str());
    for (auto data_addr : data_addrs) {
        std::vector<ea_t> xrefs = getXrefs(data_addr);

        for (auto xref : xrefs) {
            msg("[%s] findSmiHandlers: 0x%016llX\n", plugin_name, u64_addr(xref));
            auto res = findSmiHandlers(xref, prefix);
            smiHandlers.insert(smiHandlers.end(), res.begin(), res.end());
        }
    }

    return smiHandlers;
}

//----------------------------------------------------------------------
// Find SwSmiHandler function inside SMM drivers in case where
// EFI_SMM_SW_DISPATCH(2)_PROTOCOL_GUID is a local variable
std::vector<func_t *> findSmiHandlersSmmDispatchStack(std::vector<json> stackGuids,
                                                      std::string prefix) {
    // TODO: make it generic
    std::vector<func_t *> smiHandlers;

    for (auto guid : stackGuids) {
        std::string name = static_cast<std::string>(guid["name"]);

        if (name != "EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID" &&
            name != "EFI_SMM_SW_DISPATCH_PROTOCOL_GUID") {
            continue;
        }

        ea_t address = static_cast<ea_t>(guid["address"]);
```

```
        msg("[%s] found EFI_SMM_SW_DISPATCH(2)_PROTOCOL_GUID on stack: "
            "0x%016llX\n",
            plugin_name, u64_addr(address));
        auto res = findSmiHandlers(address, prefix);
        smiHandlers.insert(smiHandlers.end(), res.begin(), res.end());
    }

    return smiHandlers;
}

//-----------------------------------------------------------------------
// Find gSmmVar->SmmGetVariable calls via EFI_SMM_VARIABLE_PROTOCOL_GUID
std::vector<ea_t> findSmmGetVariableCalls(std::vector<segment_t *> dataSegments,
                                          std::vector<json> *allServices) {
    msg("[%s] gSmmVar->SmmGetVariable calls finding via "
        "EFI_SMM_VARIABLE_PROTOCOL_GUID\n",
        plugin_name);
    std::vector<ea_t> smmGetVariableCalls;
    EfiGuid guid = {0xed32d533,
                    0x99e6,
                    0x4209,
                    {0x9c, 0xc0, 0x2d, 0x72, 0xcd, 0xd9, 0x98,
                     0xa7}}; // EFI_SMM_VARIABLE_PROTOCOL_GUID

    // Find all EFI_GUID EFI_SMM_VARIABLE_PROTOCOL_GUID addresses
    std::vector<ea_t> data_addrs = findData(0, BADADDR, guid.uchar_data().data(), 16);
    std::vector<ea_t> gSmmVarAddrs; // Find all gSmmVar variables
    for (auto data_addr : data_addrs) {
        std::vector<ea_t> xrefs = getXrefs(data_addr);

        for (auto xref : xrefs) {
            segment_t *seg = getseg(static_cast<ea_t>(xref));
            qstring seg_name;
            get_segm_name(&seg_name, seg);
            msg("[%s] EFI_SMM_VARIABLE_PROTOCOL_GUID xref address: 0x%016llX, "
                "segment: %s\n",
                plugin_name, u64_addr(xref), seg_name.c_str());

            size_t index = seg_name.find(".text");
            if (index == std::string::npos) {
                continue;
            }

            insn_t insn;
            ea_t ea = xref;
            for (auto i = 0; i < 8; i++) {
                // Find `lea r8, <gSmmVar_addr>` instruction
                ea = prev_head(ea, 0);
                decode_insn(&insn, ea);
                if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
                    insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_mem) {
                    msg("[%s] gSmmVar address: 0x%016llX\n", plugin_name,
                        u64_addr(insn.ops[1].addr));
                    set_cmt(ea, "EFI_SMM_VARIABLE_PROTOCOL *gSmmVar", true);
                    setPtrTypeAndName(insn.ops[1].addr, "gSmmVar",
                                      "EFI_SMM_VARIABLE_PROTOCOL");
                    gSmmVarAddrs.push_back(insn.ops[1].addr);
                    break;
                }
            }
        }
    }

    if (!gSmmVarAddrs.size()) {
```

```
        msg("[%s] can't find gSmmVar addresses\n", plugin_name);
        return smmGetVariableCalls;
    }

    for (auto smmVarAddr : gSmmVarAddrs) {
        std::vector<ea_t> smmVarXrefs = getXrefs(static_cast<ea_t>(smmVarAddr));
        for (auto smmVarXref : smmVarXrefs) {
            segment_t *seg = getseg(static_cast<ea_t>(smmVarXref));
            qstring seg_name;
            get_segm_name(&seg_name, seg);
            msg("[%s] gSmmVar xref address: 0x%016llX, segment: %s\n", plugin_name,
                u64_addr(smmVarXref), seg_name.c_str());

            size_t index = seg_name.find(".text");
            if (index == std::string::npos) {
                continue;
            }

            uint16 gSmmVarReg = 0xffff;
            insn_t insn;
            ea_t ea = static_cast<ea_t>(smmVarXref);
            decode_insn(&insn, ea);

            if (insn.itype == NN_mov && insn.ops[0].type == o_reg &&
                insn.ops[1].type == o_mem) {
                gSmmVarReg = insn.ops[0].reg;
                for (auto i = 0; i < 16; i++) {
                    ea = next_head(ea, BADADDR);
                    decode_insn(&insn, ea);

                    if (insn.itype == NN_callni && gSmmVarReg == insn.ops[0].reg &&
                        insn.ops[0].addr == 0) {
                        msg("[%s] gSmmVar->SmmGetVariable found: 0x%016llX\n",
                            plugin_name, u64_addr(ea));

                        if (find(smmGetVariableCalls.begin(),
smmGetVariableCalls.end(),
                                 ea) == smmGetVariableCalls.end()) {
                            smmGetVariableCalls.push_back(ea);
                        }

                        // Temporarily add a "virtual" smm service call
                        // for easier annotations and UI

                        std::string cmt = getSmmVarComment();
                        set_cmt(ea, cmt.c_str(), true);
                        opStroff(ea, "EFI_SMM_VARIABLE_PROTOCOL");
                        msg("[%s] 0x%016llX : %s\n", plugin_name, u64_addr(ea),
                            "SmmGetVariable");
                        std::string smm_call = "gSmmVar->SmmGetVariable";
                        json smm_item;
                        smm_item["address"] = ea;
                        smm_item["service_name"] = smm_call;
                        smm_item["table_name"] =
                            static_cast<std::string>("EFI_SMM_VARIABLE_PROTOCOL");
                        smm_item["offset"] = 0;

                        if (find(allServices->begin(), allServices->end(), smm_item) ==
                            allServices->end()) {
                            allServices->push_back(smm_item);
                        }

                        break;
                    }
```

```
                }
            }
        }
    }
    return smmGetVariableCalls;
}

std::vector<ea_t> resolveEfiSmmCpuProtocol(std::vector<json> stackGuids,
                                           std::vector<json> dataGuids,
                                           std::vector<json> *allServices) {
    std::vector<ea_t> readSaveStateCalls;
    msg("[%s] Looking for EFI_SMM_CPU_PROTOCOL\n", plugin_name);
    std::vector<ea_t> codeAddrs;
    std::vector<ea_t> gSmmCpuAddrs;
    for (auto guid : stackGuids) {
        std::string name = static_cast<std::string>(guid["name"]);
        if (name != "EFI_SMM_CPU_PROTOCOL_GUID")
            continue;
        ea_t address = static_cast<ea_t>(guid["address"]);
        msg("[%s] found EFI_SMM_CPU_PROTOCOL on stack: 0x%016llX\n", plugin_name,
            u64_addr(address));
        codeAddrs.push_back(address);
    }

    for (auto guid : dataGuids) {
        std::string name = static_cast<std::string>(guid["name"]);
        if (name != "EFI_SMM_CPU_PROTOCOL_GUID")
            continue;

        ea_t address = static_cast<ea_t>(guid["address"]);
        msg("[%s] found EFI_SMM_CPU_PROTOCOL: 0x%016llX\n", plugin_name,
            u64_addr(address));
        std::vector<ea_t> guidXrefs = getXrefs(address);

        for (auto guidXref : guidXrefs) {
            segment_t *seg = getseg(static_cast<ea_t>(guidXref));
            qstring seg_name;
            get_segm_name(&seg_name, seg);
            size_t index = seg_name.find(".text");
            if (index == std::string::npos) {
                continue;
            }
            codeAddrs.push_back(static_cast<ea_t>(guidXref));
        }
    }

    for (auto addr : codeAddrs) {
        msg("[%s] current address: 0x%016llX\n", plugin_name, u64_addr(addr));
        insn_t insn;
        ea_t ea = prev_head(addr, 0);

        for (auto i = 0; i < 8; i++) {
            // Find 'lea r8, <gSmmCpu_addr>' instruction
            decode_insn(&insn, ea);
            if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
                insn.ops[0].reg == REG_R8 && insn.ops[1].type == o_mem) {
                msg("[%s] gSmmCpu address: 0x%016llX\n", plugin_name,
                    u64_addr(insn.ops[1].addr));
                set_cmt(ea, "EFI_SMM_CPU_PROTOCOL *gSmmCpu", true);
                setPtrTypeAndName(insn.ops[1].addr, "gSmmCpu", "EFI_SMM_CPU_PROTOCOL");
                gSmmCpuAddrs.push_back(insn.ops[1].addr);
                break;
            }
            ea = prev_head(ea, 0);
```

```
    }
}

if (!gSmmCpuAddrs.size()) {
    msg("[%s] can't find gSmmCpu addresses\n", plugin_name);
    return readSaveStateCalls;
}

for (auto smmCpu : gSmmCpuAddrs) {
    std::vector<ea_t> smmCpuXrefs = getXrefs(static_cast<ea_t>(smmCpu));

    for (auto smmCpuXref : smmCpuXrefs) {
        segment_t *seg = getseg(static_cast<ea_t>(smmCpuXref));
        qstring seg_name;
        get_segm_name(&seg_name, seg);
        size_t index = seg_name.find(".text");

        if (index == std::string::npos) {
            continue;
        }

        uint16_t gSmmCpuReg = 0xffff;
        insn_t insn;
        ea_t ea = static_cast<ea_t>(smmCpuXref);
        decode_insn(&insn, ea);

        if (insn.itype == NN_mov && insn.ops[0].type == o_reg &&
            insn.ops[1].type == o_mem) {
            gSmmCpuReg = insn.ops[0].reg;

            for (auto i = 0; i < 16; i++) {
                ea = next_head(ea, BADADDR);
                decode_insn(&insn, ea);

                if (insn.itype == NN_callni && gSmmCpuReg == insn.ops[0].reg &&
                    insn.ops[0].addr == 0) {
                    if (find(readSaveStateCalls.begin(), readSaveStateCalls.end(),
                             ea) == readSaveStateCalls.end()) {
                        readSaveStateCalls.push_back(ea);
                    }

                    opStroff(ea, "EFI_SMM_CPU_PROTOCOL");
                    msg("[%s] 0x%016llX : %s\n", plugin_name, u64_addr(ea),
                        "gSmmCpu->ReadSaveState");
                    std::string smm_call = "gSmmCpu->ReadSaveState";
                    json smm_item;
                    smm_item["address"] = ea;
                    smm_item["service_name"] = smm_call;
                    smm_item["table_name"] =
                        static_cast<std::string>("EFI_SMM_CPU_PROTOCOL");
                    smm_item["offset"] = 0;

                    if (find(allServices->begin(), allServices->end(), smm_item) ==
                        allServices->end()) {
                        allServices->push_back(smm_item);
                    }

                    break;
                }
            }
        }
    }
}
return readSaveStateCalls;
```

```
}

ea_t markChildSwSmiHandler(ea_t ea) {
    insn_t insn;
    auto addr = prev_head(ea, 0);
    decode_insn(&insn, addr);
    while (!is_basic_block_end(insn, false)) {
        // for next iteration
        decode_insn(&insn, addr);
        addr = prev_head(addr, 0);

        // check current instruction
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_RCX) {
            if (insn.ops[1].type != o_mem) {
                continue;
            }
            set_name(insn.ops[1].addr, "ChildSwSmiHandler", SN_FORCE);
            return insn.ops[1].addr;
        }
    }
    return BADADDR;
}
}
```

efiXplorer/efiSmmUtils.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiSmmUtils.h
 *
 */

#pragma once

#include "efiUtils.h"

std::vector<ea_t> findSmstSwDispatch(std::vector<ea_t> gBsList);
std::vector<ea_t> findSmstSmmBase(std::vector<ea_t> gBsList);
std::vector<func_t *> findSmiHandlers(ea_t address, std::string prefix);
std::vector<func_t *> findSmiHandlersSmmDispatch(EfiGuid guid, std::string prefix);
std::vector<func_t *> findSmiHandlersSmmDispatchStack(std::vector<json> stackGuids,
                                                      std::string prefix);
std::vector<ea_t> findSmmGetVariableCalls(std::vector<segment_t *> dataSegments,
                                          std::vector<json> *allServices);
```

```
std::vector<ea_t> resolveEfiSmmCpuProtocol(std::vector<json> stackGuids,
                                           std::vector<json> dataGuids,
                                           std::vector<json> *allServices);
ea_t markChildSwSmiHandler(ea_t ea);
}
```

efiXplorer/efiUi.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiUi.cpp
 *
 */

#include "efiUi.h"
#include "efiDeps.h"
#include "efiGlobal.h"

static const char plugin_name[] = "efiXplorer";

// vulns column widths
const int vulns_chooser_t::widths_vulns[] = {
    16, // Address
    32, // Vuln type
};

// vulns column headers
const char *const vulns_chooser_t::header_vulns[] = {
    "Address", // 0
    "Type",    // 1
};

// guids column widths
const int guids_chooser_t::widths_guids[] = {
    16, // Address
    32, // GUID
    32  // Name
};

// guids column headers
const char *const guids_chooser_t::header_guids[] = {
    "Address", // 0
    "GUID",    // 1
    "Name"     // 2
};
```

```
// protocols column widths
const int protocols_chooser_t::widths_protocols[] = {
    16, // Address
    32, // GUID
    32, // Name
    32, // Service
    32  // Module
};

// protocols column headers
const char *const protocols_chooser_t::header_protocols[] = {
    "Address", // 0
    "GUID",    // 1
    "Name",    // 2
    "Service", // 3
    "Module"   // 4
};

// services column widths
const int s_chooser_t::widths_s[] = {
    16, // Address
    32, // Service name
    32, // Table name
};

// services column headers
const char *const s_chooser_t::header_s[] = {
    "Address",      // 0
    "Service name", // 1
    "Table name"    // 2
};

// services column widths
const int nvram_chooser_t::widths_nvram[] = {
    16, // Address
    32, // Variable name
    32, // Variable GUID
    32, // Service
    64, // Attributes
};

// NVRAMs column headers
const char *const nvram_chooser_t::header_nvram[] = {
    "Address",       // 0
    "Variable name", // 1
    "Variable GUID", // 2
    "Service",       // 3
    "Attributes"     // 4
};

inline nvram_chooser_t::nvram_chooser_t(const char *title_, bool ok,
                                        std::vector<json> nvram)
    : chooser_t(0, qnumber(widths_nvram), widths_nvram, header_nvram, title_), list() {
    CASSERT(qnumber(widths_nvram) == qnumber(header_nvram));
    build_list(ok, nvram);
}

void idaapi nvram_chooser_t::get_row(qstrvec_t *cols_, int *, chooser_item_attrs_t *,
                                     size_t n) const {
    ea_t ea = list[n];
    qstrvec_t &cols = *cols_;
    json item = chooser_nvram[n];
    std::string name = static_cast<std::string>(item["VariableName"]);
```

```
    std::string guid = static_cast<std::string>(item["VendorGuid"]);
    std::string service = static_cast<std::string>(item["service"]);
    std::string attributes = static_cast<std::string>(item["AttributesHumanReadable"]);
    cols[0].sprnt("%016llX", u64_addr(ea));
    cols[1].sprnt("%s", name.c_str());
    cols[2].sprnt("%s", guid.c_str());
    cols[3].sprnt("%s", service.c_str());
    cols[4].sprnt("%s", attributes.c_str());
    CASSERT(qnumber(header_nvram) == 5);
}

inline vulns_chooser_t::vulns_chooser_t(const char *title_, bool ok,
                                        std::vector<json> vulns)
    : chooser_t(0, qnumber(widths_vulns), widths_vulns, header_vulns, title_), list() {
    CASSERT(qnumber(widths_vulns) == qnumber(header_vulns));
    build_list(ok, vulns);
}

void idaapi vulns_chooser_t::get_row(qstrvec_t *cols_, int *, chooser_item_attrs_t *,
                                     size_t n) const {
    ea_t ea = list[n];
    qstrvec_t &cols = *cols_;
    json item = chooser_vulns[n];
    std::string type = static_cast<std::string>(item["type"]);
    cols[0].sprnt("%016llX", u64_addr(ea));
    cols[1].sprnt("%s", type.c_str());
    CASSERT(qnumber(header_vulns) == 2);
}

inline guids_chooser_t::guids_chooser_t(const char *title_, bool ok,
                                        std::vector<json> guids)
    : chooser_t(0, qnumber(widths_guids), widths_guids, header_guids, title_), list() {
    CASSERT(qnumber(widths_guids) == qnumber(header_guids));
    build_list(ok, guids);
}

void idaapi guids_chooser_t::get_row(qstrvec_t *cols_, int *, chooser_item_attrs_t *,
                                     size_t n) const {
    ea_t ea = list[n];
    qstrvec_t &cols = *cols_;
    json item = chooser_guids[n];
    std::string guid = static_cast<std::string>(item["guid"]);
    std::string name = static_cast<std::string>(item["name"]);
    cols[0].sprnt("%016llX", u64_addr(ea));
    cols[1].sprnt("%s", guid.c_str());
    cols[2].sprnt("%s", name.c_str());
    CASSERT(qnumber(header_guids) == 3);
}

inline protocols_chooser_t::protocols_chooser_t(const char *title_, bool ok,
                                                std::vector<json> protocols,
                                                std::string name_key_)
    : chooser_t(0, qnumber(widths_protocols), widths_protocols,
header_protocols, title_),
      list() {
    CASSERT(qnumber(widths_protocols) == qnumber(header_protocols));
    name_key = name_key_;
    build_list(ok, protocols);
}

void idaapi protocols_chooser_t::get_row(qstrvec_t *cols_, int *,
chooser_item_attrs_t *,
                                         size_t n) const {
    ea_t ea = list[n];
```

```
    qstrvec_t &cols = *cols_;
    json item = chooser_protocols[n];
    std::string name = static_cast<std::string>(item[name_key]);
    std::string service = static_cast<std::string>(item["service"]);
    std::string protGuid = static_cast<std::string>(item["guid"]);
    std::string moduleName = static_cast<std::string>(item["module"]);
    cols[0].sprnt("%016llX", u64_addr(ea));
    cols[1].sprnt("%s", protGuid.c_str());
    cols[2].sprnt("%s", name.c_str());
    cols[3].sprnt("%s", service.c_str());
    cols[4].sprnt("%s", moduleName.c_str());
    CASSERT(qnumber(header_protocols) == 5);
}

inline s_chooser_t::s_chooser_t(const char *title_, bool ok,
std::vector<json> services)
    : chooser_t(0, qnumber(widths_s), widths_s, header_s, title_), list() {
    CASSERT(qnumber(widths_s) == qnumber(header_s));
    build_list(ok, services);
}

void idaapi s_chooser_t::get_row(qstrvec_t *cols_, int *, chooser_item_attrs_t *,
                                 size_t n) const {
    ea_t ea = list[n];
    qstrvec_t &cols = *cols_;
    json item = chooser_s[n];
    std::string service_name = static_cast<std::string>(item["service_name"]);
    std::string table_name = static_cast<std::string>(item["table_name"]);
    cols[0].sprnt("%016llX", u64_addr(ea));
    cols[1].sprnt("%s", service_name.c_str());
    cols[2].sprnt("%s", table_name.c_str());
    CASSERT(qnumber(header_s) == 3);
}

bool nvram_show(std::vector<json> nvram, qstring title) {
    bool ok;
    // open the window
    nvram_chooser_t *ch = new nvram_chooser_t(title.c_str(), ok, nvram);
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

bool vulns_show(std::vector<json> vulns, qstring title) {
    bool ok;
    // open the window
    vulns_chooser_t *ch = new vulns_chooser_t(title.c_str(), ok, vulns);
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

bool guids_show(std::vector<json> guids, qstring title) {
    bool ok;
    // open the window
    guids_chooser_t *ch = new guids_chooser_t(title.c_str(), ok, guids);
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

bool protocols_show(std::vector<json> protocols, qstring title) {
    bool ok;
    // open the window
```

```
    protocols_chooser_t *ch =
        new protocols_chooser_t(title.c_str(), ok, protocols, "prot_name");
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

bool ppis_show(std::vector<json> ppis, qstring title) {
    bool ok;
    // open the window
    protocols_chooser_t *ch =
        new protocols_chooser_t(title.c_str(), ok, ppis, "ppi_name");
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

bool services_show(std::vector<json> services, qstring title) {
    bool ok;
    // open the window
    s_chooser_t *ch = new s_chooser_t(title.c_str(), ok, services);
    // default cursor position is 0 (first row)
    ch->choose();
    return true;
}

//--------------------------------------------------------------------------
// Action handler for protocols dependencies
struct protocols_deps_handler_t : public action_handler_t {
    virtual int idaapi activate(action_activation_ctx_t *ctx) {
        auto n = ctx->chooser_selection.at(0);
        json info = g_deps.protocolsChooser[n];

        if (info.is_null()) {
            return -1; // protocol not found
        }

        // get dependencies for protocol
        std::string guid = info["guid"];
        json d = g_deps.protocolsByGuids[guid];

        // print dependencies for current
        // protocol in output window
        std::string s = d.dump(2);
        msg("[%s] dependencies for protocol with GUID %s: %s\n", plugin_name,
            guid.c_str(), s.c_str());

        return 0;
    }

    virtual action_state_t idaapi update(action_update_ctx_t *ctx) {
        return AST_ENABLE_ALWAYS;
    }
};

static protocols_deps_handler_t protocols_deps_ah;
action_desc_t protocols_deps = ACTION_DESC_LITERAL(
    "efiXplorer:protocolsDeps", "Show dependencies", &protocols_deps_ah, NULL,
NULL, -1);

void attachActionProtocolsDeps() {
    // Attach action in protocols chooser
    TWidget *widget = find_widget("efiXplorer: protocols");
    if (widget == nullptr) {
```

```
        msg("[%s] can not find efiXplorer: protocols chooser", plugin_name);
        return;
    }
    register_action(protocols_deps);
    attach_action_to_popup(widget, nullptr, protocols_deps.name);
}

//--------------------------------------------------------------------------
// Action handler for showing the sequence of modules execution
struct modules_seq_handler_t : public action_handler_t {
    virtual int idaapi activate(action_activation_ctx_t *ctx) {
        g_deps.buildModulesSequence();
        std::string s = g_deps.modulesSequence.dump(2);
        msg("[%s] sequence of modules execution: %s\n", plugin_name, s.c_str());

        return 0;
    }

    virtual action_state_t idaapi update(action_update_ctx_t *ctx) {
        return AST_ENABLE_ALWAYS;
    }
};

static modules_seq_handler_t modules_seq_ah;
action_desc_t modules_seq =
    ACTION_DESC_LITERAL("efiXplorer:modulesSeq", "Show the sequence of
modules execution",
                        &modules_seq_ah, NULL, NULL, -1);

void attachActionModulesSeq() {
    // Attach action in protocols chooser
    TWidget *widget = find_widget("efiXplorer: protocols");
    if (widget == nullptr) {
        msg("[%s] can not find efiXplorer: protocols chooser", plugin_name);
        return;
    }
    register_action(modules_seq);
    attach_action_to_popup(widget, nullptr, modules_seq.name);
}

//--------------------------------------------------------------------------
// Action handler (load efiXplorer analysis report)
struct action_handler_loadreport_t : public action_handler_t {
    virtual int idaapi activate(action_activation_ctx_t *ctx) {
        std::filesystem::path reportPath;
        char *file = ask_file(false, "*.json", "Load efiXplorer analysis report");
        if (file == nullptr) {
            msg("[%s] report file not specified\n", plugin_name);
            return -1;
        }
        reportPath /= file;
        msg("[%s] loading report from %s file\n", plugin_name, reportPath.c_str());

        json reportData;
        try {
            std::ifstream in(reportPath);
            in >> reportData;
        } catch (std::exception &e) {
            msg("[%s] report file is invalid, check its contents\n", plugin_name);
            return -1;
        }

        // Initialize vuln types list
        std::vector<std::string> vulnTypes{
```

```
    "smm_callout", "pei_get_variable_buffer_overflow",
    "get_variable_buffer_overflow", "smm_get_variable_buffer_overflow"};

// Show all choosers with data from report
qstring title;

try {
    auto protocols = reportData["allProtocols"];
    if (!protocols.is_null()) { // show protocols
        title = "efiXplorer: protocols";
        protocols_show(protocols, title);
    }
    auto ppis = reportData["allPPIs"];
    if (!ppis.is_null()) { // show PPIs
        title = "efiXplorer: PPIs";
        protocols_show(ppis, title);
    }
    auto services = reportData["allServices"];
    if (!services.is_null()) { // show services
        title = "efiXplorer: services";
        services_show(services, title);
    }
    auto guids = reportData["allGuids"];
    if (!guids.is_null()) { // show GUIDs
        title = "efiXplorer: GUIDs";
        guids_show(guids, title);
    }
    auto nvram = reportData["nvramVariables"];
    if (!nvram.is_null()) { // show NVRAM
        title = "efiXplorer: NVRAM";
        nvram_show(nvram, title);
    }
    auto vulns = reportData["vulns"];
    if (!vulns.is_null()) { // show vulns
        std::vector<json> vulnsRes;
        for (auto vulnType : vulnTypes) {
            // For each vuln type add list of vulns in `vulnsRes`
            auto vulnAddrs = vulns[vulnType];
            if (vulnAddrs.is_null()) {
                continue;
            }
            for (auto addr : vulnAddrs) {
                json item;
                item["type"] = vulnType;
                item["address"] = addr;
                vulnsRes.push_back(item);
            }
        }
        if (vulnsRes.size()) {
            title = "efiXplorer: vulns";
            vulns_show(vulnsRes, title);
        }
    }

    // Init public EdiDependencies members
    g_deps.getProtocolsChooser(protocols);
    g_deps.getProtocolsByGuids(protocols);

    // Save all protocols information to build dependencies
    attachActionProtocolsDeps();
    attachActionModulesSeq();
} catch (std::exception &e) {
    msg("[%s] report file is invalid, check its contents\n", plugin_name);
    return -1;
```

```
        }

        return 0;
    }

    virtual action_state_t idaapi update(action_update_ctx_t *ctx) {
        return AST_ENABLE_ALWAYS;
    }
};
static action_handler_loadreport_t load_report_handler;

//--------------------------------------------------------------------------
// Action to load efiXplorer analysis report
action_desc_t action_load_report =
    ACTION_DESC_LITERAL("efiXplorer:loadReport", "efiXplorer analysis report...",
                        &load_report_handler, NULL, NULL, -1);
}
```

efiXplorer/efiUi.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiUi.h
 *
 */

#pragma once

#include "efiUtils.h"

//--------------------------------------------------------------------------
// Vulns chooser
class vulns_chooser_t : public chooser_t {
  protected:
    static const int widths_vulns[];
    static const char *const header_vulns[];

  public:
    eavec_t list;
    json chooser_vulns;

    // this object must be allocated using `new`
    vulns_chooser_t(const char *title, bool ok, std::vector<json> vulns);

    // function that is used to decide whether a new chooser should be opened or
```

```
    // we can use the existing one. The contents of the window are completely
    // determined by its title
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }

    // function that returns number of lines in the list
    virtual size_t idaapi get_count() const { return list.size(); }

    // function that generates the list line
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;

    // function that is called when the user hits `Enter`
    virtual cbret_t idaapi enter(size_t n) {
        if (n < list.size())
            jumpto(list[n]);
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<json> vulns) {
        size_t n = 0;
        for (auto vuln : vulns) {
            list.push_back(vuln["address"]);
            chooser_vulns[n] = vuln;
            n++;
        }
        ok = true;
    };
};

//---------------------------------------------------------------------------
// GUIDs chooser
class guids_chooser_t : public chooser_t {
  protected:
    static const int widths_guids[];
    static const char *const header_guids[];

  public:
    eavec_t list;
    json chooser_guids;

    // this object must be allocated using `new`
    guids_chooser_t(const char *title, bool ok, std::vector<json> guids);

    // function that is used to decide whether a new chooser should be opened or
    // we can use the existing one. The contents of the window are completely
    // determined by its title
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }

    // function that returns number of lines in the list
    virtual size_t idaapi get_count() const { return list.size(); }

    // function that generates the list line
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;
```

```
    // function that is called when the user hits `Enter`
    virtual cbret_t idaapi enter(size_t n) {
        if (n < list.size())
            jumpto(list[n]);
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<json> guids) {
        size_t n = 0;
        for (auto guid : guids) {
            list.push_back(guid["address"]);
            chooser_guids[n] = guid;
            n++;
        }
        ok = true;
    };
};

//-------------------------------------------------------------------------
// Protocols chooser
class protocols_chooser_t : public chooser_t {
  protected:
    static const int widths_protocols[];
    static const char *const header_protocols[];

  public:
    eavec_t list;
    json chooser_protocols;
    std::string name_key;

    // this object must be allocated using `new`
    protocols_chooser_t(const char *title, bool ok, std::vector<json> interfaces,
                        std::string name_key);

    // function that is used to decide whether a new chooser should be opened or
    // we can use the existing one. The contents of the window are completely
    // determined by its title
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }

    // function that returns number of lines in the list
    virtual size_t idaapi get_count() const { return list.size(); }

    // function that generates the list line
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;

    // function that is called when the user hits `Enter`
    virtual cbret_t idaapi enter(size_t n) {
        if (n < list.size())
            jumpto(list[n]);
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<json> protocols) {
        size_t n = 0;
        for (auto protocol : protocols) {
            list.push_back(protocol["xref"]);
            chooser_protocols[n] = protocol;
```

```
            n++;
        }
        ok = true;
    };
};

//-------------------------------------------------------------------------
// Service chooser (address : service_name)
class s_chooser_t : public chooser_t {
  protected:
    static const int widths_s[];
    static const char *const header_s[];

  public:
    eavec_t list;
    json chooser_s;

    // this object must be allocated using `new`
    s_chooser_t(const char *title, bool ok, std::vector<json> services);

    // function that is used to decide whether a new chooser should be opened or
    // we can use the existing one. The contents of the window are completely
    // determined by its title
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }

    // function that returns number of lines in the list
    virtual size_t idaapi get_count() const { return list.size(); }

    // function that generates the list line
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;

    // function that is called when the user hits `Enter`
    virtual cbret_t idaapi enter(size_t n) {
        if (n < list.size())
            jumpto(list[n]);
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<json> services) {
        size_t n = 0;
        for (auto j_service : services) {
            list.push_back(j_service["address"]);
            chooser_s[n] = j_service;
            n++;
        }
        ok = true;
    };
};

//-------------------------------------------------------------------------
// NVRAM chooser
class nvram_chooser_t : public chooser_t {
  protected:
    static const int widths_nvram[];
    static const char *const header_nvram[];

  public:
    eavec_t list;
```

```
    json chooser_nvram;

    // this object must be allocated using `new`
    nvram_chooser_t(const char *title, bool ok, std::vector<json> nvrams);

    // function that is used to decide whether a new chooser should be opened or
    // we can use the existing one. The contents of the window are completely
    // determined by its title
    virtual const void *get_obj_id(size_t *len) const {
        *len = strlen(title);
        return title;
    }

    // function that returns number of lines in the list
    virtual size_t idaapi get_count() const { return list.size(); }

    // function that generates the list line
    virtual void idaapi get_row(qstrvec_t *cols, int *icon_,
chooser_item_attrs_t *attrs,
                                size_t n) const;

    // function that is called when the user hits `Enter`
    virtual cbret_t idaapi enter(size_t n) {
        if (n < list.size())
            jumpto(list[n]);
        return cbret_t();
    }

  protected:
    void build_list(bool ok, std::vector<json> nvrams) {
        size_t n = 0;
        for (auto nvram : nvrams) {
            list.push_back(nvram["addr"]);
            chooser_nvram[n] = nvram;
            n++;
        }
        ok = true;
    };
};

extern action_desc_t action_load_report;

bool nvram_show(std::vector<json> nvram, qstring title);
bool vulns_show(std::vector<json> vulns, qstring title);
bool guids_show(std::vector<json> guid, qstring title);
bool protocols_show(std::vector<json> protocols, qstring title);
bool ppis_show(std::vector<json> protocols, qstring title);
bool services_show(std::vector<json> services, qstring title);
void attachActionProtocolsDeps();
void attachActionModulesSeq();
}
```

efiXplorer/efiUtils.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
```

```
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiUtils.cpp
 *
 */

#include "efiUtils.h"
#include "efiGlobal.h"
#include "tables/efi_system_tables.h"

struct pei_services_entry {
    char name[256];
    uint32_t offset;
    char description[1024];
    uint32_t nr_args;
    char prototype[512];
    uint32_t count;
    uint16_t ppi_guid_push_number;
    uint16_t guid_offset;
};

// can be used after Hex-Rays based analysis
std::vector<ea_t> g_get_smst_location_calls;
std::vector<ea_t> g_smm_get_variable_calls;
std::vector<ea_t> g_smm_set_variable_calls;

extern struct pei_services_entry pei_services_table[];
extern size_t pei_services_table_size;

struct variable_ppi_entry {
    char name[256];
    uint32_t offset;
    char description[1024];
    uint32_t nr_args;
    char prototype[512];
};

extern struct variable_ppi_entry variable_ppi_table[];
extern size_t variable_ppi_table_size;

static const char plugin_name[] = "efiXplorer";

//--------------------------------------------------------------------------
// Create EFI_GUID structure
void createGuidStructure(ea_t ea) {
    static const char struct_name[] = "_EFI_GUID";
    struc_t *sptr = get_struc(get_struc_id(struct_name));
    if (sptr == nullptr) {
        sptr = get_struc(add_struc(-1, struct_name));
        if (sptr == nullptr)
            return;
        add_struc_member(sptr, "data1", -1, dword_flag(), NULL, 4);
        add_struc_member(sptr, "data2", -1, word_flag(), NULL, 2);
        add_struc_member(sptr, "data3", -1, word_flag(), NULL, 2);
        add_struc_member(sptr, "data4", -1, byte_flag(), NULL, 8);
```

```
    }
    asize_t size = get_struc_size(sptr);
    create_struct(ea, size, sptr->id);
}

//-------------------------------------------------------------------------
// Set EFI_GUID type
void setGuidType(ea_t ea) {
    tinfo_t tinfo;
    if (tinfo.get_named_type(get_idati(), "EFI_GUID")) {
        apply_tinfo(ea, tinfo, TINFO_DEFINITE);
    }
}

//-------------------------------------------------------------------------
// Set type and name
void setTypeAndName(ea_t ea, std::string name, std::string type) {
    set_name(ea, name.c_str(), SN_FORCE);
    tinfo_t tinfo;
    if (tinfo.get_named_type(get_idati(), type.c_str())) {
        apply_tinfo(ea, tinfo, TINFO_DEFINITE);
    }
}

//-------------------------------------------------------------------------
// Get file format name (fileformatname)
std::string getFileFormatName() {
    char file_format[256] = {0};
    get_file_type_name(file_format, 256);
    return static_cast<std::string>(file_format);
}

//-------------------------------------------------------------------------
// Get input file type (64-bit, 32-bit image or UEFI firmware)
uint8_t getInputFileType() {
    processor_t &ph = PH;
    auto filetype = (filetype_t)inf.filetype;
    auto bits = inf_is_64bit() ? 64 : inf_is_32bit_exactly() ? 32 : 16;

    // check if input file is UEFI firmware image
    if (getFileFormatName().find("UEFI") != std::string::npos) {
        return UEFI;
    }

    if (filetype == f_PE || filetype == f_ELF) {
        if (ph.id == PLFM_386) {
            if (bits == 64) // x86 64-bit executable
                return X64;
            if (bits == 32) // x86 32-bit executable
                return X86;
        }
        if (ph.id == PLFM_ARM) {
            if (bits == 64) // ARM 64-bit executable
                return ARM64;
        }
    }
    return UNSUPPORTED_TYPE;
}

//-------------------------------------------------------------------------
// Get input file type (PEI or DXE-like). No reliable way to determine FFS
// file type given only its PE/TE image section, so hello heuristics
uint8_t guessFileType(uint8_t arch, std::vector<json> *allGuids) {
    if (arch == UEFI) {
```

```
        return FTYPE_DXE_AND_THE_LIKE;
    }
    segment_t *hdr_seg = get_segm_by_name("HEADER");
    if (hdr_seg == NULL) {
        return FTYPE_DXE_AND_THE_LIKE;
    }
    uint64_t signature = get_wide_word(hdr_seg->start_ea);
    bool hasPeiGuids = false;
    for (auto guid = allGuids->begin(); guid != allGuids->end(); guid++) {
        json guidVal = *guid;

        if (static_cast<std::string>(guidVal["name"]).find("PEI") !=
std::string::npos ||
            static_cast<std::string>(guidVal["name"]).find("Pei") !=
std::string::npos) {
            hasPeiGuids = true;
            break;
        }
    }

    bool hasPeiInPath = false;
    char fileName[0x1000] = {0};
    get_input_file_path(fileName, sizeof(fileName));
    auto fileNameStr = static_cast<std::string>(fileName);
    if ((fileNameStr.find("Pei") != std::string::npos ||
         fileNameStr.find("pei") != std::string::npos || signature == VZ) &&
        arch == X86) {
        hasPeiInPath = true;
    }

    if (signature == VZ || hasPeiGuids) {
        msg("[%s] Parsing binary file as PEI, signature = %llx, hasPeiGuids = %d\n",
            plugin_name, signature, hasPeiGuids);
        return FTYPE_PEI;
    } else {
        msg("[%s] Parsing binary file as DXE/SMM, signature = %llx, hasPeiGuids
= %d\n",
            plugin_name, signature, hasPeiGuids);
        return FTYPE_DXE_AND_THE_LIKE;
    }
}

uint8_t getFileType(std::vector<json> *allGuids) {
    uint8_t arch = getInputFileType();
    if (arch == UEFI || arch == X64) {
        return FTYPE_DXE_AND_THE_LIKE;
    }
    auto ftype = guessFileType(arch, allGuids);
    auto deflt = ftype == FTYPE_DXE_AND_THE_LIKE;
    auto fmt_param = ftype == FTYPE_DXE_AND_THE_LIKE ? "DXE/SMM" : "PEI";
    auto btnId = ask_buttons("DXE/SMM", "PEI", "", deflt, "Parse file as
%s", fmt_param);
    if (btnId == ASKBTN_YES) {
        return FTYPE_DXE_AND_THE_LIKE;
    } else {
        return FTYPE_PEI;
    }
}

//-------------------------------------------------------------------------
// Get boot service description comment
std::string getBsComment(uint32_t offset, uint8_t arch) {
    uint32_t offset_current;
    std::string cmt = "gBS->";
```

```
    for (auto i = 0; i < BTABLE_LEN; i++) {
        offset_current = boot_services_table[i].offset64;
        if (arch == X86) {
            offset_current = boot_services_table[i].offset32;
        }
        if (offset == offset_current) {
            cmt += static_cast<std::string>(boot_services_table[i].name) + "()\n" +
                   static_cast<std::string>(boot_services_table[i].prototype) + "\n" +
                   static_cast<std::string>(boot_services_table[i].parameters);
            break;
        }
    }
    return cmt;
}

//-------------------------------------------------------------------------
// Get Pei service description comment (X86 is assumed)
std::string getPeiSvcComment(uint32_t offset) {
    std::string cmt = "gPS->";
    for (auto i = 0; i < pei_services_table_size; i++) {
        if (offset == pei_services_table[i].offset) {
            cmt += static_cast<std::string>(pei_services_table[i].name) + "()\n" +
                   static_cast<std::string>(pei_services_table[i].prototype);
            break;
        }
    }
    return cmt;
}

//-------------------------------------------------------------------------
// Get PPI service description comment (X86 is assumed)
std::string getPPICallComment(uint32_t offset, std::string name) {
    std::string cmt = name + "->"; // VariablePpi
    for (auto i = 0; i < variable_ppi_table_size; i++) {
        if (offset == variable_ppi_table[i].offset) {
            cmt += static_cast<std::string>(variable_ppi_table[i].name) + "()\n" +
                   static_cast<std::string>(variable_ppi_table[i].prototype);
            break;
        }
    }
    return cmt;
}

//-------------------------------------------------------------------------
// Get SMM service description comment
std::string getSmmVarComment() {
    std::string name = "EFI_SMM_VARIABLE_PROTOCOL";
    std::string prototype = "EFI_STATUS (EFIAPI *EFI_GET_VARIABLE)"
                            "(IN CHAR16 *VariableName, "
                            "IN EFI_GUID *VendorGuid, "
                            "OUT UINT32 *Attributes, OPTIONAL "
                            "IN OUT UINTN *DataSize, "
                            "OUT VOID *Data OPTIONAL);";

    std::string cmt = name + "->SmmGetVariable()\n" + prototype;
    return cmt;
}

//-------------------------------------------------------------------------
// Get runtime service description comment
std::string getRtComment(uint32_t offset, uint8_t arch) {
    ea_t offset_arch;
    std::string cmt = "gRT->";
    for (auto i = 0; i < RTABLE_LEN; i++) {
```

```
        offset_arch = runtime_services_table[i].offset64;
        if (arch == X86) {
            offset_arch = runtime_services_table[i].offset32;
        }
        if (offset == offset_arch) {
            cmt += static_cast<std::string>(runtime_services_table[i].name) + "()\n" +
                   static_cast<std::string>(runtime_services_table[i].prototype) +
"\n" +
                   static_cast<std::string>(runtime_services_table[i].parameters);
            break;
        }
    }
    return cmt;
}

//------------------------------------------------------------------------
// Find address of global gBS var for X64 module for each service
ea_t findUnknownBsVarX64(ea_t ea) {
    ea_t resAddr = 0;
    insn_t insn;

    // Check 10 instructions below
    for (int i = 0; i < 10; i++) {
        decode_insn(&insn, ea);
        if (insn.itype == NN_mov && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_RAX && insn.ops[1].type == o_mem) {
            resAddr = insn.ops[1].addr;
            break;
        }
        ea = prev_head(ea, 0);
    }
    return resAddr;
}

//------------------------------------------------------------------------
// Get all xrefs for given address
std::vector<ea_t> getXrefs(ea_t addr) {
    std::vector<ea_t> xrefs;
    ea_t xref = get_first_dref_to(addr);
    while (xref != BADADDR) {
        xrefs.push_back(xref);
        xref = get_next_dref_to(addr, xref);
    }
    return xrefs;
}

//------------------------------------------------------------------------
// Get all xrefs for given array element
std::vector<ea_t> getXrefsToArray(ea_t addr) {
    ea_t first_ea;
    ea_t ea = addr;
    while (true) {
        auto ptr = get_qword(ea);
        auto xrefs = getXrefs(ptr);
        if (std::find(xrefs.begin(), xrefs.end(), ea) == xrefs.end()) {
            break;
        }
        first_ea = ea;
        ea -= 8;
    }
    return getXrefs(first_ea);
}

//------------------------------------------------------------------------
```

```
// Wrapper for op_stroff function
bool opStroff(ea_t addr, std::string type) {
    insn_t insn;
    decode_insn(&insn, addr);
    tid_t struc_id = get_struc_id(type.c_str());
    return op_stroff(insn, 0, &struc_id, 1, 0);
}

//--------------------------------------------------------------------------
// Get pointer to named type and apply it
bool setPtrType(ea_t addr, std::string type) {
    tinfo_t tinfo;
    if (!tinfo.get_named_type(get_idati(), type.c_str())) {
        return false;
    }
    tinfo_t ptrTinfo;
    ptrTinfo.create_ptr(tinfo);
    apply_tinfo(addr, ptrTinfo, TINFO_DEFINITE);
    return true;
}

//--------------------------------------------------------------------------
// Set name and apply pointer to named type
void setPtrTypeAndName(ea_t ea, std::string name, std::string type) {
    set_name(ea, name.c_str(), SN_FORCE);
    setPtrType(ea, type.c_str());
}

//--------------------------------------------------------------------------
// Get guids.json file name
std::filesystem::path getGuidsJsonFile() {
    std::filesystem::path guidsJsonPath;

    // check {idadir}/plugins/guids.json
    guidsJsonPath /= idadir("plugins");
    guidsJsonPath /= "guids.json";
    if (std::filesystem::exists(guidsJsonPath)) {
        return guidsJsonPath;
    }

    // check {idadir}/plugins/guids/guids.json
    guidsJsonPath.clear();
    guidsJsonPath /= idadir("plugins");
    guidsJsonPath /= "guids";
    guidsJsonPath /= "guids.json";
    if (std::filesystem::exists(guidsJsonPath)) {
        return guidsJsonPath;
    }

    // Try to load it from the per-user directory.
    guidsJsonPath.clear();
    guidsJsonPath /= get_user_idadir();
    guidsJsonPath /= "plugins";
    guidsJsonPath /= "guids.json";
    if (std::filesystem::exists(guidsJsonPath)) {
        return guidsJsonPath;
    }

    guidsJsonPath.clear();
    guidsJsonPath /= get_user_idadir();
    guidsJsonPath /= "plugins";
    guidsJsonPath /= "guids";
    guidsJsonPath /= "guids.json";
    if (std::filesystem::exists(guidsJsonPath)) {
```

```
        return guidsJsonPath;
    }

    // Does not exist.
    guidsJsonPath.clear();
    return guidsJsonPath;
}

//--------------------------------------------------------------------------
// Get json summary file name
std::filesystem::path getSummaryFile() {
    std::string idbPath;
    idbPath = get_path(PATH_TYPE_IDB);
    std::filesystem::path logFile;
    logFile /= idbPath;
    logFile.replace_extension(".json");
    return logFile;
}

//--------------------------------------------------------------------------
// Check for summary json file exist
bool summaryJsonExist() {
    std::string idbPath;
    idbPath = get_path(PATH_TYPE_IDB);
    std::filesystem::path logFile;
    logFile /= idbPath;
    logFile.replace_extension(".json");
    return std::filesystem::exists(logFile);
}

//--------------------------------------------------------------------------
// Change EFI_SYSTEM_TABLE *SystemTable to EFI_PEI_SERVICES **PeiService
// for ModuleEntryPoint
void setEntryArgToPeiSvc() {
    for (auto idx = 0; idx < get_entry_qty(); idx++) {
        uval_t ord = get_entry_ordinal(idx);
        ea_t start_ea = get_entry(ord);
        tinfo_t tif_ea;
        if (guess_tinfo(&tif_ea, start_ea) == GUESS_FUNC_FAILED) {
            msg("[%s] guess_tinfo failed, start_ea = 0x%016llX, idx=%d\n", plugin_name,
                u64_addr(start_ea), idx);
            continue;
        }
        func_type_data_t funcdata;
        if (!tif_ea.get_func_details(&funcdata)) {
            msg("[%s] get_func_details failed, %d\n", plugin_name, idx);
            continue;
        }
        tinfo_t tif_pei;
        bool res = tif_pei.get_named_type(get_idati(), "EFI_PEI_SERVICES");
        if (!res) {
            msg("[%s] get_named_type failed, res = %d, idx=%d\n", plugin_name,
res, idx);
            continue;
        }
        tinfo_t ptrTinfo;
        tinfo_t ptrPtrTinfo;
        ptrTinfo.create_ptr(tif_pei);
        ptrPtrTinfo.create_ptr(ptrTinfo);
        // funcdata.size() does not work for aarch64
        if (funcdata.size() == 2) {
            funcdata[1].type = ptrPtrTinfo;
            funcdata[1].name = "PeiServices";
            tinfo_t func_tinfo;
```

```
            if (!func_tinfo.create_func(funcdata)) {
                msg("[%s] create_func failed, idx=%d\n", plugin_name, idx);
                continue;
            }
            if (!apply_tinfo(start_ea, func_tinfo, TINFO_DEFINITE)) {
                msg("[%s] apply_tinfo failed, idx=%d\n", plugin_name, idx);
                continue;
            }
        }
    }
}

bool setRetToPeiSvc(ea_t start_ea) {
    tinfo_t tif_ea;
    if (guess_tinfo(&tif_ea, start_ea) == GUESS_FUNC_FAILED) {
        msg("[%s] guess_tinfo failed, function = 0x%016llX", plugin_name,
            u64_addr(start_ea));
        return false;
    }
    func_type_data_t fi;
    if (!tif_ea.get_func_details(&fi)) {
        msg("[%s] get_func_details failed, function = 0x%016llX", plugin_name,
            u64_addr(start_ea));
        return false;
    }
    tinfo_t tif_pei;
    bool res = tif_pei.get_named_type(get_idati(), "EFI_PEI_SERVICES");
    if (!res) {
        msg("[%s] get_named_type failed, res = %d\n", plugin_name, res);
        return false;
    }
    tinfo_t ptrTinfo;
    tinfo_t ptrPtrTinfo;
    ptrTinfo.create_ptr(tif_pei);
    ptrPtrTinfo.create_ptr(ptrTinfo);

    fi.rettype = ptrPtrTinfo;

    tinfo_t func_tinfo;
    if (!func_tinfo.create_func(fi)) {
        msg("[%s] create_func failed, function = 0x%016llX", plugin_name,
            u64_addr(start_ea));
        return false;
    }
    if (!apply_tinfo(start_ea, func_tinfo, TINFO_DEFINITE)) {
        msg("[%s] apply_tinfo failed, function = 0x%016llX", plugin_name,
            u64_addr(start_ea));
        return false;
    }
    return true;
}

//--------------------------------------------------------------------------
// Add EFI_PEI_SERVICES_4 structure
bool addStrucForShiftedPtr() {
    auto sid = add_struc(BADADDR, "EFI_PEI_SERVICES_4");
    if (sid == BADADDR) {
        return false;
    }

    auto new_struct = get_struc(sid);
    if (new_struct == nullptr) {
        return false;
    }
```

```
    add_struc_member(new_struct, nullptr, 0, dword_flag(), nullptr, 4);
    add_struc_member(new_struct, nullptr, 4, dword_flag(), nullptr, 4);
    set_member_name(new_struct, 0, "PeiServices");
    set_member_name(new_struct, 4, "PeiServices4");

    tinfo_t tinfo;
    if (!tinfo.get_named_type(get_idati(), "EFI_PEI_SERVICES")) {
        return false;
    }

    // set type "EFI_PEI_SERVICES **PeiServices"
    tinfo_t ptrTinfo;
    tinfo_t ptr2Tinfo;
    ptrTinfo.create_ptr(tinfo);
    ptr2Tinfo.create_ptr(ptrTinfo);

    auto member = get_member_by_name(new_struct, "PeiServices");
    set_member_tinfo(new_struct, member, 0, ptr2Tinfo, 0);

    return true;
}

//-------------------------------------------------------------------------
// Change the value of a number to match the data type
uval_t truncImmToDtype(uval_t value, op_dtype_t dtype) {
    switch (dtype) {
    case dt_byte:
        return value & 0xff;
    case dt_word:
        return value & 0xffff;
    case dt_dword:
        return value & 0xffffffff;
    default:
        return value;
    }
}

//-------------------------------------------------------------------------
// Get module name by address
qstring getModuleNameLoader(ea_t address) {
    segment_t *seg = getseg(address);
    qstring seg_name;
    get_segm_name(&seg_name, seg);
    return seg_name.remove(seg_name.size() - 7, seg_name.size());
}

//-------------------------------------------------------------------------
// Get GUID data by address
json getGuidByAddr(ea_t addr) {
    return json::array(
        {get_wide_dword(addr), get_wide_word(addr + 4), get_wide_word(addr + 6),
         get_wide_byte(addr + 8), get_wide_byte(addr + 9), get_wide_byte(addr + 10),
         get_wide_byte(addr + 11), get_wide_byte(addr + 12), get_wide_byte(addr + 13),
         get_wide_byte(addr + 14), get_wide_byte(addr + 15)});
}

//-------------------------------------------------------------------------
// Validate GUID value
bool checkGuid(json guid) {
    if (static_cast<uint32_t>(guid[0]) == 0x00000000 && (uint16_t)guid[1] == 0x0000) {
        return false;
    }
    if (static_cast<uint32_t>(guid[0]) == 0xffffffff && (uint16_t)guid[1] == 0xffff) {
```

```
        return false;
    }
    return true;
}

//--------------------------------------------------------------------------
// Convert GUID value to string
std::string getGuidFromValue(json guid) {
    char guidStr[37] = {0};
    snprintf(guidStr, 37, "%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X%02X",
             static_cast<uint32_t>(guid[0]), static_cast<uint16_t>(guid[1]),
             static_cast<uint16_t>(guid[2]), static_cast<uint8_t>(guid[3]),
             static_cast<uint8_t>(guid[4]), static_cast<uint8_t>(guid[5]),
             static_cast<uint8_t>(guid[6]), static_cast<uint8_t>(guid[7]),
             static_cast<uint8_t>(guid[8]), static_cast<uint8_t>(guid[9]),
             static_cast<uint8_t>(guid[10]));
    return static_cast<std::string>(guidStr);
}

std::vector<uint8_t> unpackGuid(std::string guid) {
    std::vector<uint8_t> res;
    std::string delimiter = "-";
    std::string byte_str;
    uint8_t byte;
    size_t pos = 0;

    auto index = 0;
    while ((pos = guid.find(delimiter)) != std::string::npos) {
        std::vector<uint8_t> tmp;
        auto hex = guid.substr(0, pos);
        if (hex.size() % 2) {
            break;
        }
        for (auto i = 0; i < hex.size(); i += 2) {
            byte_str = hex.substr(i, 2);
            byte = static_cast<uint8_t>(strtol(byte_str.c_str(), NULL, 16));
            tmp.push_back(byte);
        }
        if (index != 3) {
            res.insert(res.end(), tmp.rbegin(), tmp.rend());
        } else {
            res.insert(res.end(), tmp.begin(), tmp.end());
        }
        index += 1;
        guid.erase(0, pos + delimiter.size());
        tmp.clear();
    }

    for (auto i = 0; i < guid.size(); i += 2) {
        byte_str = guid.substr(i, 2);
        byte = static_cast<uint8_t>(strtol(byte_str.c_str(), NULL, 16));
        res.push_back(byte);
    }

    return res;
}

std::vector<ea_t> searchProtocol(std::string protocol) {
    uchar bytes[17] = {0};
    std::vector<ea_t> res;
    auto guid_bytes = unpackGuid(protocol);
    std::copy(guid_bytes.begin(), guid_bytes.end(), bytes);
    ea_t start = 0;
    while (true) {
```

```
        ea_t addr = bin_search2(start, BADADDR, bytes, nullptr,
16, BIN_SEARCH_FORWARD);
        if (addr == BADADDR) {
            break;
        }
        res.push_back(addr);
        start = addr + 16;
    }
    return res;
}

bool checkInstallProtocol(ea_t ea) {
    insn_t insn;
    // search for `call [REG + offset]` insn
    // offset in [0x80, 0xA8, 0x148]
    ea_t addr = ea;
    for (auto i = 0; i < 16; i++) {
        addr = next_head(addr, BADADDR);
        decode_insn(&insn, addr);
        if ((insn.itype == NN_jmpni || insn.itype == NN_callni) &&
            insn.ops[0].type == o_displ) {
            auto service = insn.ops[0].addr;
            // check for InstallProtocolInterface, InstallMultipleProtocolInterfaces,
            // SmmInstallProtocolInterface
            if (service == 0x80 || service == 0xa8 || service == 0x148) {
                return true;
            }
        }
    }
    return false;
}

//------------------------------------------------------------------------
// Convert 64-bit value to hex string
std::string getHex(uint64_t value) {
    char hexstr[21] = {};
    snprintf(hexstr, 21, "%llX", value);
    return static_cast<std::string>(hexstr);
}

//------------------------------------------------------------------------
// Make sure the first argument looks like protocol
bool bootServiceProtCheck(ea_t callAddr) {
    bool valid = false;
    insn_t insn;
    auto addr = prev_head(callAddr, 0);
    decode_insn(&insn, addr);
    while (!is_basic_block_end(insn, false)) {

        // for next iteration
        decode_insn(&insn, addr);
        addr = prev_head(addr, 0);

        // check current instruction
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_RCX) {
            if (insn.ops[1].type == o_mem) {
                // will still be a false positive if the Handle in
                // SmmInstallProtocolInterface is a global variable)
                valid = true;
            }
            break;
        }
    }
```

```
    return valid;
}

//-------------------------------------------------------------------------
// Make sure that the address does not belong to the protocol interface
bool bootServiceProtCheckXrefs(ea_t callAddr) {
    insn_t insn;
    for (auto xref : getXrefs(callAddr)) {
        decode_insn(&insn, xref);
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_R8) {
            // load interface instruction
            return false;
        }
    }
    return true;
}

bool markCopy(ea_t codeAddr, ea_t varAddr, std::string type) {
    insn_t insn;
    int reg = -1;
    ea_t ea = codeAddr;
    ea_t varCopy = BADADDR;
    decode_insn(&insn, ea);

    // get `reg` value
    if (insn.itype == NN_mov && insn.ops[0].type == o_reg && insn.ops[1].type ==
o_mem &&
        insn.ops[1].addr == varAddr) {
        reg = insn.ops[0].reg;
    }

    if (reg == -1) {
        return false;
    }

    for (auto i = 0; i < 8; ++i) {
        ea = next_head(ea, BADADDR);
        decode_insn(&insn, ea);

        if (is_basic_block_end(insn, false)) {
            break;
        }

        if ((insn.itype == NN_callni || insn.itype == NN_call) ||
            (insn.ops[0].type == o_reg && insn.ops[0].reg == reg)) {
            break;
        }

        // get `varCopy`
        if (insn.itype == NN_mov && insn.ops[0].type == o_mem &&
            insn.ops[1].type == o_reg && insn.ops[1].reg == reg) {
            varCopy = insn.ops[0].addr;
            msg("[efiXplorer] Found copy for global variable:
0x%016llX\n", u64_addr(ea));
            break;
        }
    }

    if (varCopy == BADADDR) {
        return false;
    }

    std::string name;
```

```
    if (type == "gSmst") {
        setPtrTypeAndName(varCopy, "gSmst", "_EFI_SMM_SYSTEM_TABLE2");
    }

    if (type == "gBS") {
        setPtrTypeAndName(varCopy, "gBS", "EFI_BOOT_SERVICES");
    }

    if (type == "gRT") {
        setPtrTypeAndName(varCopy, "gRT", "EFI_RUNTIME_SERVICES");
    }

    return true;
}

bool markCopiesForGlobalVars(std::vector<ea_t> globalVars, std::string type) {
    for (auto var : globalVars) {
        auto xrefs = getXrefs(var);
        for (auto addr : xrefs) {
            markCopy(addr, var, type);
        }
    }
    return true;
}

//--------------------------------------------------------------------------
// Generate name string from type
std::string typeToName(std::string type) {
    std::string result;
    size_t counter = 0;
    for (char const &c : type) {
        if ((c >= 'a' && c <= 'z') || (c >= '0' && c <= '9')) {
            result.push_back(c);
            counter += 1;
            continue;
        }

        if (c >= 'A' && c <= 'Z') {
            if (counter > 0) {
                result.push_back(c + 32);
            } else
                result.push_back(c);
            counter += 1;
            continue;
        }

        if (c == '_') {
            counter = 0;
        } else {
            counter += 1;
        }
    }
    return result;
}

xreflist_t xrefsToStackVar(ea_t funcEa, qstring varName) {
    struc_t *frame = get_frame(funcEa);
    func_t *func = get_func(funcEa);
    member_t member; // Get member by name
    bool found = false;
    for (int i = 0; i < frame->memqty; i++) {
        member = frame->members[i];
        qstring name;
```

```
        get_member_name(&name, frame->members[i].id);
        if (name == varName) {
            found = true;
            break;
        }
    }
    xreflist_t xrefs_list; // Get xrefs
    if (found) {
        build_stkvar_xrefs(&xrefs_list, func, &member);
    }
    return xrefs_list;
}

void opstroffForAddress(ea_t ea, qstring typeName) {
    insn_t insn;

    for (auto i = 0; i < 16; i++) {
        ea = next_head(ea, BADADDR);
        decode_insn(&insn, ea);
        // Found interface function call
        if ((insn.itype == NN_call || insn.itype == NN_callfi ||
             insn.itype == NN_callni) &&
            (insn.ops[0].type == o_displ || insn.ops[0].type == o_phrase) &&
            insn.ops[0].reg == REG_RAX) {
            opStroff(ea, static_cast<std::string>(typeName.c_str()));
            msg("[%s] Mark arguments at address 0x%016llX (interface type: %s)\n",
                plugin_name, u64_addr(ea), typeName.c_str());

            // check for EfiSmmBase2Protocol->GetSmstLocation
            if (typeName == qstring("EFI_SMM_BASE2_PROTOCOL") &&
                insn.ops[0].type == o_displ && insn.ops[0].addr == 8) {
                if (!addrInVec(g_get_smst_location_calls, ea)) {
                    g_get_smst_location_calls.push_back(ea);
                }
            }

            if (typeName == qstring("EFI_SMM_VARIABLE_PROTOCOL") &&
                insn.ops[0].type == o_phrase) {
                if (!addrInVec(g_smm_get_variable_calls, ea)) {
                    g_smm_get_variable_calls.push_back(ea);
                }
            }

            if (typeName == qstring("EFI_SMM_VARIABLE_PROTOCOL") &&
                insn.ops[0].type == o_displ && insn.ops[0].addr == 0x10) {
                if (!addrInVec(g_smm_set_variable_calls, ea)) {
                    g_smm_set_variable_calls.push_back(ea);
                }
            }

            break;
        }
        // If the RAX value is overridden
        if (insn.ops[0].reg == REG_RAX) {
            break;
        }
    }
}

//--------------------------------------------------------------------------
// Mark the arguments of each function from an interface derived from
// a local variable
void opstroffForInterface(xreflist_t localXrefs, qstring typeName) {
    insn_t insn;
```

```
    for (auto xref : localXrefs) {
        decode_insn(&insn, xref.ea);
        if (insn.itype == NN_mov && insn.ops[0].reg == REG_RAX) {
            opstroffForAddress(xref.ea, typeName);
        }
    }
}

//--------------------------------------------------------------------------
// Mark the arguments of each function from an interface derived from
// a global variable
void opstroffForGlobalInterface(std::vector<ea_t> xrefs, qstring typeName) {
    insn_t insn;
    for (auto ea : xrefs) {
        decode_insn(&insn, ea);
        if (insn.itype == NN_mov && insn.ops[0].reg == REG_RAX) {
            opstroffForAddress(ea, typeName);
        }
    }
}

bool qwordInVec(std::vector<uint64_t> vec, uint64_t value) {
    return find(vec.begin(), vec.end(), value) != vec.end();
}

bool addrInVec(std::vector<ea_t> vec, ea_t addr) {
    return find(vec.begin(), vec.end(), addr) != vec.end();
}

bool jsonInVec(std::vector<json> vec, json item) {
    return find(vec.begin(), vec.end(), item) != vec.end();
}

bool addrInTables(std::vector<ea_t> gStList, std::vector<ea_t> gBsList,
                  std::vector<ea_t> gRtList, ea_t ea) {
    return (addrInVec(gStList, ea) || addrInVec(gBsList, ea) ||
addrInVec(gRtList, ea));
}

std::vector<ea_t> findData(ea_t start_ea, ea_t end_ea, uchar *data, size_t len) {
    std::vector<ea_t> res;
    ea_t start = start_ea;
    int counter = 0;
    while (true) {
        auto ea = bin_search2(start, end_ea, data, nullptr, len, BIN_SEARCH_FORWARD);
        if (ea == BADADDR) {
            break;
        }
        res.push_back(ea);
        start = ea + len;
    }
    return res;
}

//--------------------------------------------------------------------------
// Get wide string by address
std::string getWideString(ea_t addr) {
    std::string res;
    int index = 0;
    while (get_wide_word(addr + index)) {
        auto byte = get_wide_byte(addr + index);
        if (byte < 0x20 || byte > 0x7e) {
            return "INVALID_STRING";
        }
```

```
        res.push_back(byte);
        index += 2;
    }
    return res;
}

//-------------------------------------------------------------------------
// Get EfiGuid by address
EfiGuid getGlobalGuid(ea_t addr) {
    EfiGuid guid;
    guid.data1 = get_wide_dword(addr);
    guid.data2 = get_wide_word(addr + 4);
    guid.data3 = get_wide_word(addr + 6);
    for (auto i = 0; i < 8; i++) {
        guid.data4[i] = static_cast<uint8_t>(get_wide_byte(addr + 8 + i));
    }
    return guid;
}

//-------------------------------------------------------------------------
// Get EfiGuid by stack offset
EfiGuid getStackGuid(func_t *f, uint64_t offset) {
    EfiGuid guid;
    insn_t insn;
    auto ea = f->start_ea;
    int counter = 0;
    while (ea <= f->end_ea) {
        if (counter == 16) {
            break;
        }
        ea = next_head(ea, BADADDR);
        decode_insn(&insn, ea);
        if (insn.itype == NN_mov && insn.ops[0].type == o_displ &&
            (insn.ops[0].reg == REG_RSP || insn.ops[0].reg == REG_RBP) &&
            insn.ops[1].type == o_imm) {
            if (insn.ops[0].addr == offset) {
                guid.data1 = insn.ops[1].value;
                counter += 4;
                continue;
            }
            if (insn.ops[0].addr == offset + 4) {
                guid.data2 = insn.ops[1].value & 0xffff;
                guid.data3 = (insn.ops[1].value >> 16) & 0xffff;
                counter += 4;
                continue;
            }
            if (insn.ops[0].addr == offset + 8) {
                auto dword = insn.ops[1].value;
                guid.data4[0] = dword & 0xff;
                guid.data4[1] = (dword >> 8) & 0xff;
                guid.data4[2] = (dword >> 16) & 0xff;
                guid.data4[3] = (dword >> 24) & 0xff;
                counter += 4;
                continue;
            }
            if (insn.ops[0].addr == offset + 12) {
                auto dword = insn.ops[1].value;
                guid.data4[4] = dword & 0xff;
                guid.data4[5] = (dword >> 8) & 0xff;
                guid.data4[6] = (dword >> 16) & 0xff;
                guid.data4[7] = (dword >> 24) & 0xff;
                counter += 4;
                continue;
            }
```

```
        }
    }
    return guid;
}

std::string getTable(std::string service_name) {
    for (auto i = 0; i < BTABLE_LEN; i++) {
        if (static_cast<std::string>(boot_services_table[i].name) == service_name) {
            return "EFI_BOOT_SERVICES";
        }
    }
    for (auto i = 0; i < RTABLE_LEN; i++) {
        if (static_cast<std::string>(runtime_services_table[i].name) == service_name) {
            return "EFI_RUNTIME_SERVICES";
        }
    }
    return "OTHER";
}

std::string lookupBootServiceName(uint64_t offset) {
    for (auto i = 0; i < BTABLE_LEN; i++) {
        if (boot_services_table[i].offset64 == offset) {
            return static_cast<std::string>(boot_services_table[i].name);
        }
    }
    return "Unknown";
}

std::string lookupRuntimeServiceName(uint64_t offset) {
    for (auto i = 0; i < RTABLE_LEN; i++) {
        if (runtime_services_table[i].offset64 == offset) {
            return static_cast<std::string>(runtime_services_table[i].name);
        }
    }
    return "Unknown";
}

uint64_t u64_addr(ea_t addr) { return static_cast<uint64_t>(addr); }

uint32_t u32_addr(ea_t addr) { return static_cast<uint32_t>(addr); }

uint16_t get_machine_type() {
    ea_t pe_offset = get_dword(0x3c);
    return get_word(pe_offset + 4);
}
}
```

efiXplorer/efiUtils.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiUtils.h
 *
 */

#pragma once

// 3rd party
#include "json.hpp"

#include <algorithm>
#include <allins.hpp>
#include <auto.hpp>
#include <bytes.hpp>
#include <diskio.hpp>
#include <entry.hpp>
#include <filesystem>
#include <frame.hpp>
#include <fstream>
#include <graph.hpp>
#include <ida.hpp>
#include <idp.hpp>
#include <iostream>
#include <kernwin.hpp>
#include <lines.hpp>
#include <loader.hpp>
#include <name.hpp>
#include <pro.h>
#include <stdio.h>
#include <string>
#include <struct.hpp>
#include <typeinf.hpp>

#ifdef HEX_RAYS
#include <hexrays.hpp>
#endif

using namespace nlohmann;

#define BTOA(x) ((x) ? "true" : "false")

#define VZ 0x5A56
#define MZ 0x5A4D

enum MachineType { AMD64 = 0x8664, I386 = 0x014C, AARCH64 = 0xaa64 };

enum ArchFileType { UNSUPPORTED_TYPE, X86, X64, UEFI, ARM64 };

enum FfsFileType { FTYPE_PEI = 6, FTYPE_DXE_AND_THE_LIKE = 7 };

enum BootServicesOffset { BS_OFFSET_64BIT = 0x60, BS_OFFSET_32BIT = 0x3c };

enum RuntimeServiesOffset { RT_OFFSET_64BIT = 0x58, RT_OFFSET_32BIT = 0x38 };

enum Registers32bit {
    REG_EAX,
    REG_ECX,
    REG_EDX,
    REG_EBX,
```

```
    REG_ESP,
    REG_EBP,
    REG_ESI,
    REG_EDI,
    REG_AL = 0x10,
    REG_DL = 0x12
};

enum Registers64bit {
    REG_RAX,
    REG_RCX,
    REG_RDX,
    REG_RBX,
    REG_RSP,
    REG_RBP,
    REG_RSI,
    REG_RDI,
    REG_R8,
    REG_R9,
    REG_R10,
    REG_R11,
    REG_R12,
    REG_R13,
    REG_R14,
};

enum RegistersAarch64 {
    REG_C0 = 0,
    REG_C13 = 13,
    REG_X0 = 129,
    REG_X1,
    REG_X2,
    REG_X3,
    REG_X4,
    REG_X5,
    REG_X6,
    REG_X7,
    REG_X8,
    REG_X9,
    REG_X10,
    REG_X11,
    REG_X12,
    REG_X13,
    REG_X14,
    REG_X15,
    REG_X16,
    REG_X17,
    REG_X18,
    REG_X19,
    REG_X20,
    REG_X21,
    REG_X22,
    REG_X23,
    REG_X24,
    REG_X25,
    REG_X26,
    REG_X27,
    REG_X28,
    REG_X29,
    REG_X30,
    REG_XZR,
    REG_XSP,
    REG_XPC,
};
```

```
enum HelperValues {
    GUID_OFFSET_DWORD = 4,
    GUID_OFFSET_NONE = 0xffff,
    PUSH_NONE = 0xffff,
    BAD_REG = 0xffff,
};

struct EfiGuid {
    uint32_t data1;
    uint16_t data2;
    uint16_t data3;
    uint8_t data4[8];
    std::vector<uchar> uchar_data() {
        std::vector<uchar> res;
        res.push_back(data1 & 0xff);
        res.push_back(data1 >> 8 & 0xff);
        res.push_back(data1 >> 16 & 0xff);
        res.push_back(data1 >> 24 & 0xff);
        res.push_back(data2 & 0xff);
        res.push_back(data2 >> 8 & 0xff);
        res.push_back(data3 & 0xff);
        res.push_back(data3 >> 8 & 0xff);
        for (auto i = 0; i < 8; i++) {
            res.push_back(data4[i]);
        }
        return res;
    }
    std::string to_string() {
        char res[37] = {0};
        snprintf(res, 37, "%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X%02X", data1,
                 data2, data3, data4[0], data4[1], data4[2], data4[3],
data4[4], data4[5],
                 data4[6], data4[7]);
        return static_cast<std::string>(res);
    }
};

// Get input file type
// (64-bit, 32-bit image or UEFI firmware)
uint8_t getInputFileType();

// Get image type (PEI or DXE-like)
uint8_t getFileType(std::vector<json> *allGuids);

// Set EFI_GUID type
void setGuidType(ea_t ea);

// Get all data xrefs for address
std::vector<ea_t> getXrefs(ea_t addr);
std::vector<ea_t> getXrefsToArray(ea_t addr);

// Wrapper for op_stroff function
bool opStroff(ea_t addr, std::string type);

// Create EFI_GUID structure
void createGuidStructure(ea_t ea);

// Get boot service description comment
std::string getBsComment(uint32_t offset, uint8_t arch);

// Get PEI service description comment (X86 is assumed)
std::string getPeiSvcComment(uint32_t offset);
```

```
// Get PPI service description comment (X86 is assumed)
std::string getPPICallComment(uint32_t offset, std::string name);

// Get SMM service description comment
std::string getSmmVarComment();

// Get runtime service description comment
std::string getRtComment(uint32_t offset, uint8_t arch);

// Find address of global gBS variable
// for X64 module for each service
ea_t findUnknownBsVarX64(ea_t ea);

// Get pointer to named type and apply it
bool setPtrType(ea_t addr, std::string type);

// Set name and apply pointer to named type
void setPtrTypeAndName(ea_t ea, std::string name, std::string type);

// Get guids.json file name
std::filesystem::path getGuidsJsonFile();

// Get json summary file name
std::filesystem::path getSummaryFile();

// Check for summary json file exist
bool summaryJsonExist();

// Change EFI_SYSTEM_TABLE *SystemTable to EFI_PEI_SERVICES **PeiService
// for ModuleEntryPoint
void setEntryArgToPeiSvc();

// Set return value type to EFI_PEI_SERVICES **PeiService
// for specified function
bool setRetToPeiSvc(ea_t start_ea);

// Set type and name
void setTypeAndName(ea_t ea, std::string name, std::string type);

// Get module name by address
qstring getModuleNameLoader(ea_t address);

// Print std::vector<json> object
void printVectorJson(std::vector<json> in);

// Change the value of a number to match the data type
uval_t truncImmToDtype(uval_t value, op_dtype_t dtype);

// Get GUID data by address
json getGuidByAddr(ea_t addr);

// Validate GUID value
bool checkGuid(json guid);

// Make sure the first argument looks like protocol
bool bootServiceProtCheck(ea_t callAddr);

// Make sure that the address does not belong to the protocol interface
bool bootServiceProtCheckXrefs(ea_t callAddr);

// Convert GUID value to string
std::string getGuidFromValue(json guid);

// Convert string GUID to vector of bytes
```

```
std::vector<uint8_t> unpackGuid(std::string guid);

// Convert 64-bit value to hex string
std::string getHex(uint64_t value);

// Mark copies for global variables
bool markCopiesForGlobalVars(std::vector<ea_t> globalVars, std::string type);

//  Generate name string from type
std::string typeToName(std::string type);

// Get XREFs to stack variable
xreflist_t xrefsToStackVar(ea_t funcEa, qstring varName);

// Mark the arguments of each function from an interface derived from a local variable
void opstroffForInterface(xreflist_t localXrefs, qstring typeName);

// Mark the arguments of each function from an interface derived from a global variable
void opstroffForGlobalInterface(std::vector<ea_t> xrefs, qstring typeName);

// Find wrappers
bool qwordInVec(std::vector<uint64_t> vec, uint64_t value);
bool addrInVec(std::vector<ea_t> vec, ea_t addr);
bool jsonInVec(std::vector<json> vec, json item);
bool addrInTables(std::vector<ea_t> gStList, std::vector<ea_t> gBsList,
                  std::vector<ea_t> gRtList, ea_t ea);

// Search protocol GUID bytes in binary
std::vector<ea_t> searchProtocol(std::string protocol);

bool checkInstallProtocol(ea_t ea);
std::vector<ea_t> findData(ea_t start_ea, ea_t end_ea, uchar *data, size_t len);
std::string getWideString(ea_t addr);
EfiGuid getGlobalGuid(ea_t addr);
EfiGuid getStackGuid(func_t *f, uint64_t offset);
bool addStrucForShiftedPtr();
std::string getTable(std::string service_name);
std::string lookupBootServiceName(uint64_t offset);
std::string lookupRuntimeServiceName(uint64_t offset);
uint64_t u64_addr(ea_t addr);
uint32_t u32_addr(ea_t addr);
uint16_t get_machine_type();
}
```

efiXplorer/efiXplorer.cpp

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiXplorer.cpp
 *
 */

#include "efiXplorer.h"
#include "efiAnalyzer.h"
#include "efiGlobal.h"
#include "efiUi.h"

static const char plugin_name[] = "efiXplorer";
static const char plugin_hotkey[] = "Ctrl+Alt+E";
static const char plugin_comment[] =
    "This plugin performs automatic analysis of the input UEFI module";
static const char plugin_help[] =
    "This plugin performs automatic analysis of the input UEFI module";
static const char welcome_msg[] = "      ____ _  __     __\n"
                                  " ___ / _(_) |/_/__  / /__  _______ ____\n"
                                  "/ -_) _/ />  </ _ \\/ / _ \\/ __/ -_) __/\n"
                                  "\\__/_//_/_/|_/ .__/_/\\___/_/  \\__/_/\n"
                                  "            /_/\n";

// Default arguments
struct args g_args = {/* module_type */ DXE_SMM, /* disable_ui */ 0,
                      /* disable_vuln_hunt */ 0};

#if IDA_SDK_VERSION < 760
hexdsp_t *hexdsp = nullptr;
#endif

//--------------------------------------------------------------------------
static plugmod_t *idaapi init(void) {
    uint8_t file_type = getInputFileType();
    if (file_type == UNSUPPORTED_TYPE) {
        return PLUGIN_SKIP;
    }

    msg(welcome_msg);
    msg("%s\n\n", COPYRIGHT);

    // Register action
    register_action(action_load_report);
    attach_action_to_menu("File/Load file/", action_load_report.name, SETMENU_APP);

    return PLUGIN_KEEP;
}

//--------------------------------------------------------------------------
bool idaapi run(size_t arg) {

    if (arg >> 0 & 1) { // Parse arg value:
        //  * arg = 0 (000): default (DXE)
        //  * arg = 1 (001): default (PEI, 32-bit binaries only)
        //  * arg = 2 (010): disable_ui (DXE)
        //  * arg = 3 (011): disable_ui (PEI, 32-bit binaries only)
        //  * arg = 4 (100): disable_vuln_hunt (DXE)
        //  * arg = 5 (101): disable_vuln_hunt (PEI, 32-bit binaries only)
        //  * arg = 6 (110): disable_ui & disable_vuln_hunt (DXE)
        //  * arg = 7 (111): disable_ui & disable_vuln_hunt (PEI, 32-bit binaries only)
        g_args.module_type = PEI;
    }
```

```
    if (arg >> 1 & 1) {
        g_args.disable_ui = 1;
    }
    if (arg >> 2 & 1) {
        g_args.disable_vuln_hunt = 1;
    }

    msg("[%s] plugin run with argument %lu\n", plugin_name, arg);
    msg("[%s] disable_ui = %d, disable_vuln_hunt = %d\n",
plugin_name, g_args.disable_ui,
        g_args.disable_vuln_hunt);

    auto guids_path = getGuidsJsonFile();
    msg("[%s] guids.json exists: %s\n", plugin_name, BTOA(!guids_path.empty()));

    if (guids_path.empty()) {
        std::string msg_text =
            "guids.json file not found, copy \"guids\" directory to <IDA_DIR>/plugins";
        msg("[%s] %s\n", plugin_name, msg_text.c_str());
        warning("%s: %s\n", plugin_name, msg_text.c_str());
        return false;
    }

    uint8_t arch = getInputFileType();
    if (arch == X64) {
        msg("[%s] input file is 64-bit module (x86)\n", plugin_name);
        EfiAnalysis::efiAnalyzerMainX64();
    } else if (arch == X86) {
        msg("[%s] input file is 32-bit module (x86)\n", plugin_name);
        EfiAnalysis::efiAnalyzerMainX86();
    } else if (arch == UEFI) {
        msg("[%s] input file is UEFI firmware\n", plugin_name);
        warning("%s: analysis may take some time, please wait for it to complete\n",
                plugin_name);
        if (get_machine_type() == AARCH64) {
            msg("[%s] analyze AARCH64 modules\n", plugin_name);
            EfiAnalysis::efiAnalyzerMainArm();
        } else {
            msg("[%s] analyze AMD64 modules\n", plugin_name);
            EfiAnalysis::efiAnalyzerMainX64();
        }
    } else if (arch == ARM64) {
        msg("[%s] input file is 64-bit module (ARM)\n", plugin_name);
        EfiAnalysis::efiAnalyzerMainArm();
    }

    // Reset arguments
    g_args = {DXE_SMM, 0, 0};

    return true;
}

//--------------------------------------------------------------------------
// PLUGIN DESCRIPTION BLOCK
plugin_t PLUGIN = {
    IDP_INTERFACE_VERSION,
    0,              // plugin flags
    init,           // initialize plugin
    nullptr,        // terminate plugin
    run,            // invoke plugin
    plugin_comment, // long comment about the plugin
    plugin_help,    // multiline help about the plugin
    plugin_name,    // the preferred short name of the plugin
    plugin_hotkey   // the preferred hotkey to run the plugin
```

```
};
}
```

efiXplorer/efiXplorer.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2023 Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <https://www.gnu.org/licenses/>.
 *
 * efiXplorer.h
 *
 */

#pragma once

#include "efiUtils.h"

#define COPYRIGHT "(c) 2020-2023, Binarly - https://github.com/binarly-io/efiXplorer"
}
```

Chapter 2.1.0

efiXplorer/tables

efiXplorer/tables/efi_pei_tables.h

```
/*
 * ________________________.___
 * \_   _____/\_   _____/|   |
 *  |    __)_  |    __)  |   |
 *  |        \ |     \   |   |
 * /_______  / \___  /   |___|
 *         \/      \/
 *   _________       .__                 ____  __.      .__  _____
 *  /   _____/_  _  _|__| ______ _____  |    |/ _| ____ |__|/ ____\____
 *  \_____  \\ \/ \/ /  |/  ___//  ___/ |      <  /    \|  \   __\/ __ \
```

```
 *  /       \\     /|  |\___ \ \___ \  |    |  \|   |  \  ||  | \  ___/
 * /_______  / \/\_/ |__/____  >____  > |____|__ \___|  /__||__|  \___  >
 *         \/               \/     \/          \/    \/             \/
 *
 * EFI Swiss Knife
 * An IDA plugin to improve (U)EFI reversing
 *
 * Copyright (C) 2016, 2017  Pedro Vilaça (fG!) - reverser@put.as -
 * https://reverse.put.as
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * efi_system_tables.h
 *
 */

#pragma once

#include "stdint.h"

struct pei_services_entry {
    char name[256];
    uint32_t offset;
    char description[1024];
    uint32_t nr_args;
    char prototype[512];
    uint32_t count;
    // arg number with Ppi guid or PUSH_NONE if no Ppi guid applicable
    uint16_t ppi_guid_push_number;
    // GUID_OFFSET_NONE if the argument is Guid (or if no Guid arg exists)
    // Otherwise, if the arg is EFI_PEI_PPI_DESCRIPTOR/EFI_PEI_NOTIFY_DESCRIPTOR,
    // offset of Guid field inside a given structure (in bytes).
    uint16_t guid_offset;
};

struct pei_services_entry pei_services_table[] = {
    {"InstallPpi", 0x18,
     "This service is the first one provided by the PEI Foundation. This "
     "function installs an interface in the PEI PPI database by GUID. The "
     "purpose of the service is to publish an interface that other parties can "
     "use to call additional PEIMs.",
     2,
     "EFI_STATUS(EFIAPI * EFI_PEI_INSTALL_PPI) (IN CONST EFI_PEI_SERVICES "
     "**PeiServices, IN CONST EFI_PEI_PPI_DESCRIPTOR *PpiList)",
     0, 2, GUID_OFFSET_DWORD},
    {"ReInstallPpi", 0x1C,
     "This function reinstalls an interface in the PEI PPI database by GUID. "
     "The purpose of the service is to publish an interface that other parties "
     "can use to replace a same-named interface in the protocol database with "
     "a different interface.",
     3,
     "EFI_STATUS(EFIAPI * EFI_PEI_REINSTALL_PPI) (IN CONST EFI_PEI_SERVICES "
     "**PeiServices, IN CONST EFI_PEI_PPI_DESCRIPTOR *OldPpi, IN CONST "
```

```
      "EFI_PEI_PPI_DESCRIPTOR *NewPpi)",
      0, 3, GUID_OFFSET_DWORD},
     {"LocatePpi", 0x20,
      "This function locates an interface in the PEI PPI database by GUID.", 5,
      "EFI_STATUS(EFIAPI * EFI_PEI_LOCATE_PPI) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, IN CONST EFI_GUID *Guid, IN UINTN Instance, IN OUT "
      "EFI_PEI_PPI_DESCRIPTOR **PpiDescriptor OPTIONAL, IN OUT VOID **Ppi)",
      0, 2, GUID_OFFSET_NONE},
     {"NotifyPpi", 0x24,
      "This function installs a notification service to be called back when a "
      "given interface is installed or reinstalled. The purpose of the service "
      "is to publish an interface that other parties can use to call additional "
      "PPIs that may materialize later.",
      2,
      "EFI_STATUS(EFIAPI * EFI_PEI_NOTIFY_PPI) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, IN CONST EFI_PEI_NOTIFY_DESCRIPTOR *NotifyList)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"GetBootMode", 0x28, "This function returns the present value of the boot
mode.", 2,
      "EFI_STATUS(EFIAPI * EFI_PEI_GET_BOOT_MODE) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, OUT EFI_BOOT_MODE *BootMode)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"SetBootMode", 0x2C, "This function sets the value of the boot mode.", 2,
      "EFI_STATUS(EFIAPI * EFI_PEI_SET_BOOT_MODE) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, IN EFI_BOOT_MODE BootMode)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"GetHobList", 0x30,
      "This function returns the pointer to the list of Hand-Off Blocks (HOBs) "
      "in memory.",
      2,
      "EFI_STATUS(EFIAPI * EFI_PEI_GET_HOB_LIST) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, OUT VOID **HobList)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"CreateHob", 0x34,
      "This service, published by the PEI Foundation, abstracts the creation of "
      "a Hand-Off Block's (HOB's) headers.",
      4,
      "EFI_STATUS(EFIAPI * EFI_PEI_CREATE_HOB) (IN CONST EFI_PEI_SERVICES "
      "**PeiServices, IN UINT16 Type, IN UINT16 Length, IN OUT VOID **Hob)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"FfsFindNextVolume", 0x38,
      "The purpose of the service is to abstract the capability of the PEI "
      "Foundation to discover instances of firmware volumes in the system. "
      "Given the input file pointer, this service searches for the next "
      "matching file in the Firmware File System (FFS) volume.",
      3,
      "EFI_STATUS (EFIAPI *EFI_PEI_FFS_FIND_NEXT_VOLUME) (IN struct "
      "_EFI_PEI_SERVICES **PeiServices, IN UINTN Instance, IN OUT "
      "EFI_FIRMWARE_VOLUME_HEADER **FwVolHeader)",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"FfsFindNextFile", 0x3C,
      "The purpose of the service is to abstract the capability of the PEI "
      "Foundation to discover instances of firmware files in the system. Given "
      "the input file pointer, this service searches for the next matching file "
      "in the Firmware File System (FFS) volume.",
      4,
      "EFI_STATUS (EFIAPI *EFI_PEI_FFS_FIND_NEXT_FILE) (IN struct "
      "_EFI_PEI_SERVICES **PeiServices, IN EFI_FV_FILETYPE SearchType, IN "
      "EFI_FIRMWARE_VOLUME_HEADER *FwVolHeader, IN OUT EFI_FFS_FILE_HEADER "
      "**FileHeader);",
      0, PUSH_NONE, GUID_OFFSET_NONE},
     {"FfsFindSectionData", 0x40,
      "Given the input file pointer, this service searches for the next "
      "matching file in the Firmware File System (FFS) volume.",
```

```
     4,
     "EFI_STATUS (EFIAPI *EFI_PEI_FFS_FIND_SECTION_DATA) (IN struct "
     "_EFI_PEI_SERVICES **PeiServices, IN EFI_SECTION_TYPE SectionType, IN "
     "EFI_FFS_FILE_HEADER *FfsFileHeader, IN OUT VOID **SectionData);",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"InstallPeiMemory", 0x44,
     "This function registers the found memory configuration with the PEI "
     "Foundation.",
     3,
     "EFI_STATUS(EFIAPI * EFI_PEI_INSTALL_PEI_MEMORY) (IN CONST "
     "EFI_PEI_SERVICES **PeiServices, IN EFI_PHYSICAL_ADDRESS MemoryBegin, IN "
     "UINT64 MemoryLength)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"AllocatePages", 0x48,
     "The purpose of the service is to publish an interface that allows PEIMs "
     "to allocate memory ranges that are managed by the PEI Foundation.",
     4,
     "EFI_STATUS(EFIAPI * EFI_PEI_ALLOCATE_PAGES) (IN CONST EFI_PEI_SERVICES "
     "**PeiServices, IN EFI_MEMORY_TYPE MemoryType, IN UINTN Pages, OUT "
     "EFI_PHYSICAL_ADDRESS *Memory)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"AllocatePool", 0x4C,
     "The purpose of this service is to publish an interface that allows PEIMs "
     "to allocate memory ranges that are managed by the PEI Foundation.",
     3,
     " EFI_STATUS(EFIAPI * EFI_PEI_ALLOCATE_POOL) (IN CONST EFI_PEI_SERVICES "
     "**PeiServices, IN UINTN Size, OUT VOID **Buffer)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"CopyMem", 0x50, "This service copies the contents of one buffer to
another buffer.",
     3,
     "VOID(EFIAPI * EFI_PEI_COPY_MEM) (IN VOID *Destination, IN VOID *Source, "
     "IN UINTN Length)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"SetMem", 0x54, "The service fills a buffer with a specified value.", 3,
     "VOID(EFIAPI * EFI_PEI_SET_MEM) (IN VOID *Buffer, IN UINTN Size, IN UINT8 "
     "Value)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"ReportStatusCode", 0x58,
     "This service publishes an interface that allows PEIMs to report status codes. \
        ReportStatusCode() is called by PEIMs that wish to report status information on
their progress. The principal use model is for a PEIM to emit one of the standard 32-
bit error codes. This will allow a platform owner to ascertain the state of the system,
especially under conditions where the full consoles might not have been installed.",
     6,
     "EFI_STATUS(EFIAPI * EFI_PEI_REPORT_STATUS_CODE) (IN CONST "
     "EFI_PEI_SERVICES **PeiServices, IN EFI_STATUS_CODE_TYPE Type, IN "
     "EFI_STATUS_CODE_VALUE Value, IN UINT32 Instance, IN CONST EFI_GUID "
     "*CallerId OPTIONAL, IN CONST EFI_STATUS_CODE_DATA *Data OPTIONAL)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"ResetSystem", 0x5C, "Resets the entire platform. \
        This service resets the entire platform, including all processors and devices,
and reboots the system. This service will never return EFI_SUCCESS.",
     1,
     "EFI_STATUS(EFIAPI * EFI_PEI_RESET_SYSTEM) (IN CONST EFI_PEI_SERVICES "
     "**PeiServices)",
     0, PUSH_NONE, GUID_OFFSET_NONE},
    {"CpuIo", 0x60,
     "Provides an interface that a PEIM can call to execute an I/O "
     "transaction. This service is installed by an architectural PEI driver by "
     "copying the interface pointer into this table.",
     1, "", 0, PUSH_NONE, GUID_OFFSET_NONE},
    {"PciCfg", 0x64,
     "Provides an interface that a PEIM can call to execute PCI Configuration "
```

```
     "transactions. This service is installed by an architectural PEI driver "
     "by copying the interface pointer into this table.",
     1, "", 0, PUSH_NONE, GUID_OFFSET_NONE}};

size_t pei_services_table_size = sizeof(pei_services_table)
/ sizeof(pei_services_entry);

char variable_ppi_name[] = "VariablePPI";
struct variable_ppi_entry {
    char name[256];
    uint32_t offset;
    char description[1024];
    uint32_t nr_args;
    char prototype[512];
};

struct variable_ppi_entry variable_ppi_table[] = {
    {"GetVariable", 0x0,
     "This service retrieves a variable's value using its name and GUID."
     "Read the specified variable from the UEFI variable store. If the Data"
     "buffer is too small to hold the contents of the variable,"
     "the error EFI_BUFFER_TOO_SMALL is returned and DataSize is set to the"
     "required buffer size to obtain the data.",
     6,
     "EFI_STATUS (EFIAPI *EFI_PEI_GET_VARIABLE2)"
     "(IN CONSTEFI_PEI_READ_ONLY_VARIABLE2_PPI *This, "
     "IN CONST CHAR16 *VariableName, IN CONST EFI_GUID *VariableGuid,"
     "OUT UINT32 *Attributes, IN OUT UINTN *DataSize, OUT VOID *Data OPTIONAL);"},
    {"NextVariableName", 0x4,
     "This function is called multiple times to retrieve the VariableName"
     "and VariableGuid of all variables currently available in the system."
     "On each call, the previous results are passed into the interface, "
     "and, on return, the interface returns the data for the next "
     "interface. When the entire variable list has been returned, "
     "EFI_NOT_FOUND is returned.",
     4,
     "EFI_STATUS (EFIAPI *EFI_PEI_GET_NEXT_VARIABLE_NAME2)"
     "(IN CONST EFI_PEI_READ_ONLY_VARIABLE2_PPI *This,"
     "IN OUT UINTN *VariableNameSize, IN OUT CHAR16 *VariableName,"
     "IN OUT EFI_GUID *VariableGuid);"}};

size_t variable_ppi_table_size = sizeof(variable_ppi_table)
/ sizeof(variable_ppi_entry);
}
```

efiXplorer/tables/efi_services.h

```
/*
 * efiXplorer
 * Copyright (C) 2020-2022  Binarly
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <https://www.gnu.org/licenses/>.
 *
 * efi_services.h
 *
 */

#pragma once

#include "stdint.h"

enum BootServicesOffsets64bit {
    RaiseTPLOffset64 = 0x18,
    RestoreTPLOffset64 = 0x20,
    AllocatePagesOffset64 = 0x28,
    FreePagesOffset64 = 0x30,
    GetMemoryMapOffset64 = 0x38,
    AllocatePoolOffset64 = 0x40,
    FreePoolOffset64 = 0x48,
    CreateEventOffset64 = 0x50,
    SetTimerOffset64 = 0x58,
    WaitForEventOffset64 = 0x60,
    SignalEventOffset64 = 0x68,
    CloseEventOffset64 = 0x70,
    CheckEventOffset64 = 0x78,
    InstallProtocolInterfaceOffset64 = 0x80,
    RenstallProtocolInterfaceOffset64 = 0x88,
    UninstallProtocolInterfaceOffset64 = 0x90,
    HandleProtocolOffset64 = 0x98,
    RegisterProtocolNotifyOffset64 = 0xa8,
    LocateHandleOffset64 = 0xb0,
    LocateDevicePathOffset64 = 0xb8,
    InstallConfigurationTableOffset64 = 0xc0,
    LoadImageOffset64 = 0xc8,
    StartImageOffset64 = 0xd0,
    ExitOffset64 = 0xd8,
    UnloadImageOffset64 = 0xe0,
    ExitBootServicesOffset64 = 0xe8,
    GetNextMonotonicCountOffset64 = 0xf0,
    StallOffset64 = 0xf0,
    SetWatchdogTimerOffset64 = 0x100,
    ConnectControllerOffset64 = 0x108,
    DisconnectControllerOffset64 = 0x110,
    OpenProtocolOffset64 = 0x118,
    CloseProtocolOffset64 = 0x120,
    OpenProtocolInformationOffset64 = 0x128,
    ProtocolsPerHandleOffset64 = 0x130,
    LocateHandleBufferOffset64 = 0x138,
    LocateProtocolOffset64 = 0x140,
    InstallMultipleProtocolInterfacesOffset64 = 0x148,
    UninstallMultipleProtocolInterfacesOffset64 = 0x150,
    CalculateCrc32Offset64 = 0x158,
    CopyMemOffset64 = 0x160,
    SetMemOffset64 = 0x168,
    CreateEventExOffset64 = 0x170,
};

enum BootServicesOffsets32bit {
    RaiseTPLOffset32 = 0x18,
    RestoreTPLOffset32 = 0x1c,
    AllocatePagesOffset32 = 0x20,
    FreePagesOffset32 = 0x24,
    GetMemoryMapOffset32 = 0x28,
```

```
    AllocatePoolOffset32 = 0x2c,
    FreePoolOffset32 = 0x30,
    CreateEventOffset32 = 0x34,
    SetTimerOffset32 = 0x38,
    WaitForEventOffset32 = 0x3c,
    SignalEventOffset32 = 0x40,
    CloseEventOffset32 = 0x44,
    CheckEventOffset32 = 0x48,
    InstallProtocolInterfaceOffset32 = 0x4c,
    RenstallProtocolInterfaceOffset32 = 0x50,
    UninstallProtocolInterfaceOffset32 = 0x54,
    HandleProtocolOffset32 = 0x58,
    RegisterProtocolNotifyOffset32 = 0x60,
    LocateHandleOffset32 = 0x64,
    LocateDevicePathOffset32 = 0x68,
    InstallConfigurationTableOffset32 = 0x6c,
    LoadImageOffset32 = 0x70,
    StartImageOffset32 = 0x74,
    ExitOffset32 = 0x78,
    UnloadImageOffset32 = 0x7c,
    ExitBootServicesOffset32 = 0x80,
    GetNextMonotonicCountOffset32 = 0x84,
    StallOffset32 = 0x88,
    SetWatchdogTimerOffset32 = 0x8c,
    ConnectControllerOffset32 = 0x90,
    DisconnectControllerOffset32 = 0x94,
    OpenProtocolOffset32 = 0x98,
    CloseProtocolOffset32 = 0x9c,
    OpenProtocolInformationOffset32 = 0xa0,
    ProtocolsPerHandleOffset32 = 0xa4,
    LocateHandleBufferOffset32 = 0xa8,
    LocateProtocolOffset32 = 0xac,
    InstallMultipleProtocolInterfacesOffset32 = 0xb0,
    UninstallMultipleProtocolInterfacesOffset32 = 0xb4,
    CalculateCrc32Offset32 = 0xb8,
    CopyMemOffset32 = 0xbc,
    SetMemOffset32 = 0xc0,
    CreateEventExOffset32 = 0xc4,
};

enum RuntimeServicesOffsets64bit {
    GetTimeOffset64 = 0x18,
    SetTimeOffset64 = 0x20,
    GetWakeupTimeOffset64 = 0x28,
    SetWakeupTimeOffset64 = 0x30,
    SetVirtualAddressMapOffset64 = 0x38,
    ConvertPointerOffset64 = 0x40,
    GetVariableOffset64 = 0x48,
    GetNextVariableNameOffset64 = 0x50,
    SetVariableOffset64 = 0x58,
    GetNextHighMonotonicCountOffset64 = 0x60,
    ResetSystemOffset64 = 0x68,
    UpdateCapsuleOffset64 = 0x70,
    QueryCapsuleCapabilitiesOffset64 = 0x78,
    QueryVariableInfoOffset64 = 0x80,
};

enum RuntimeServicesOffsets32bit {
    GetTimeOffset32 = 0x18,
    SetTimeOffset32 = 0x1c,
    GetWakeupTimeOffset32 = 0x20,
    SetWakeupTimeOffset32 = 0x24,
    SetVirtualAddressMapOffset32 = 0x28,
    ConvertPointerOffset32 = 0x2c,
```

```
    GetVariableOffset32 = 0x30,
    GetNextVariableNameOffset32 = 0x34,
    SetVariableOffset32 = 0x38,
    GetNextHighMonotonicCountOffset32 = 0x3c,
    ResetSystemOffset32 = 0x40,
    UpdateCapsuleOffset32 = 0x44,
    QueryCapsuleCapabilitiesOffset32 = 0x48,
    QueryVariableInfoOffset32 = 0x4c,
};

enum SmmServicesOffsets64bit {
    SmmInstallConfigurationTableOffset64 = 0x28,
    SmmAllocatePoolOffset64 = 0x50,
    SmmFreePoolOffset64 = 0x58,
    SmmAllocatePagesOffset64 = 0x60,
    SmmFreePagesOffset64 = 0x68,
    SmmStartupThisApOffset64 = 0x70,
    SmmInstallProtocolInterfaceOffset64 = 0xa8,
    SmmUninstallProtocolInterfaceOffset64 = 0xb0,
    SmmHandleProtocolOffset64 = 0xb8,
    SmmRegisterProtocolNotifyOffset64 = 0xc0,
    SmmLocateHandleOffset64 = 0xc8,
    SmmLocateProtocolOffset64 = 0xd0,
    SmiManageOffset64 = 0xd8,
    SmiHandlerRegisterOffset64 = 0xe0,
    SmiHandlerUnRegisterOffset64 = 0xe8,
};

enum SmmServicesOffsets32bit {
    SmmInstallConfigurationTableOffset32 = 0x20,
    SmmAllocatePoolOffset32 = 0x34,
    SmmFreePoolOffset32 = 0x38,
    SmmAllocatePagesOffset32 = 0x3c,
    SmmFreePagesOffset32 = 0x40,
    SmmStartupThisApOffset32 = 0x44,
    SmmInstallProtocolInterfaceOffset32 = 0x60,
    SmmUninstallProtocolInterfaceOffset32 = 0x64,
    SmmHandleProtocolOffset32 = 0x68,
    SmmRegisterProtocolNotifyOffset32 = 0x6c,
    SmmLocateHandleOffset32 = 0x70,
    SmmLocateProtocolOffset32 = 0x74,
    SmiManageOffset32 = 0x78,
    SmiHandlerRegisterOffset32 = 0x7c,
    SmiHandlerUnRegisterOffset32 = 0x80,
};

struct service_info_64bit {
    char service_name[64];
    uint32_t offset;
    uint32_t reg;
    uint16_t arg_index;
};

struct service_info_32bit {
    char service_name[64];
    uint32_t offset;
    uint16_t push_number;
};

struct service {
    char service_name[64];
    uint32_t offset64;
    uint32_t offset32;
};
```

```
struct service_info_64bit bootServicesTable64[] = {
    {"InstallProtocolInterface", InstallProtocolInterfaceOffset64, REG_RDX, 1},
    {"ReinstallProtocolInterface", RenstallProtocolInterfaceOffset64, REG_RDX, 1},
    {"UninstallProtocolInterface", UninstallProtocolInterfaceOffset64, REG_RDX, 1},
    {"HandleProtocol", HandleProtocolOffset64, REG_RDX, 1},
    {"RegisterProtocolNotify", RegisterProtocolNotifyOffset64, REG_RCX, 0},
    {"OpenProtocol", OpenProtocolOffset64, REG_RDX, 1},
    {"CloseProtocol", CloseProtocolOffset64, REG_RDX, 1},
    {"ProtocolsPerHandle", ProtocolsPerHandleOffset64, REG_RDX, 1},
    {"OpenProtocolInformation", OpenProtocolInformationOffset64, REG_RDX, 1},
    {"LocateHandleBuffer", LocateHandleBufferOffset64, REG_RDX, 1},
    {"LocateProtocol", LocateProtocolOffset64, REG_RCX, 0},
    {"InstallMultipleProtocolInterfaces", InstallMultipleProtocolInterfacesOffset64,
     REG_RDX, 1},
    {"UninstallMultipleProtocolInterfaces",
UninstallMultipleProtocolInterfacesOffset64,
     REG_RDX, 1}};
size_t bootServicesTable64Length =
    sizeof(bootServicesTable64) / sizeof(service_info_64bit);

struct service_info_32bit bootServicesTable32[] = {
    {"InstallProtocolInterface", InstallProtocolInterfaceOffset32, 2},
    {"ReinstallProtocolInterface", RenstallProtocolInterfaceOffset32, 2},
    {"UninstallProtocolInterface", UninstallProtocolInterfaceOffset32, 2},
    {"HandleProtocol", HandleProtocolOffset32, 2},
    {"RegisterProtocolNotify", RegisterProtocolNotifyOffset32, 1},
    {"OpenProtocol", OpenProtocolOffset32, 2},
    {"CloseProtocol", CloseProtocolOffset32, 2},
    {"ProtocolsPerHandle", ProtocolsPerHandleOffset32, 2},
    {"OpenProtocolInformation", OpenProtocolInformationOffset32, 2},
    {"LocateHandleBuffer", LocateHandleBufferOffset32, 2},
    {"LocateProtocol", LocateProtocolOffset32, 1},
    {"InstallMultipleProtocolInterfaces", InstallMultipleProtocolInterfacesOffset32,
2},
    {"UninstallMultipleProtocolInterfaces",
UninstallMultipleProtocolInterfacesOffset32,
     2}};
size_t bootServicesTable32Length =
    sizeof(bootServicesTable32) / sizeof(service_info_32bit);

struct service bootServicesTableAll[] = {
    // difficult to check false positives
    // {"RaiseTPL", RaiseTPLOffset64, RaiseTPLOffset32},
    // {"RestoreTPL", RestoreTPLOffset64, RestoreTPLOffset32},
    {"AllocatePages", AllocatePagesOffset64, AllocatePagesOffset32},
    {"FreePages", FreePagesOffset64, FreePagesOffset32},
    {"GetMemoryMap", GetMemoryMapOffset64, GetMemoryMapOffset32},
    {"AllocatePool", AllocatePoolOffset64, AllocatePoolOffset32},
    {"FreePool", FreePoolOffset64, FreePoolOffset32},
    {"CreateEvent", CreateEventOffset64, CreateEventOffset32},
    {"SetTimer", SetTimerOffset64, SetTimerOffset32},
    {"WaitForEvent", WaitForEventOffset64, WaitForEventOffset32},
    {"SignalEvent", SignalEventOffset64, SignalEventOffset32},
    {"CloseEvent", CloseEventOffset64, CloseEventOffset32},
    {"CheckEvent", CheckEventOffset64, CheckEventOffset32},
    {"InstallProtocolInterface", InstallProtocolInterfaceOffset64,
     InstallProtocolInterfaceOffset32},
    {"ReinstallProtocolInterface", RenstallProtocolInterfaceOffset64,
     RenstallProtocolInterfaceOffset32},
    {"UninstallProtocolInterface", UninstallProtocolInterfaceOffset64,
     UninstallProtocolInterfaceOffset32},
    {"HandleProtocol", HandleProtocolOffset64, HandleProtocolOffset32},
    {"RegisterProtocolNotify", RegisterProtocolNotifyOffset64,
```

```
     RegisterProtocolNotifyOffset32},
    {"LocateHandle", LocateHandleOffset64, LocateHandleOffset32},
    {"LocateDevicePath", LocateDevicePathOffset64, LocateDevicePathOffset32},
    {"InstallConfigurationTable", InstallConfigurationTableOffset64,
     InstallConfigurationTableOffset32},
    {"LoadImage", LoadImageOffset64, LoadImageOffset32},
    {"StartImage", StartImageOffset64, StartImageOffset32},
    {"Exit", ExitOffset64, ExitOffset32},
    {"UnloadImage", UnloadImageOffset64, UnloadImageOffset32},
    {"ExitBootServices", ExitBootServicesOffset64, ExitBootServicesOffset32},
    {"GetNextMonotonicCount", GetNextMonotonicCountOffset64,
     GetNextMonotonicCountOffset32},
    {"Stall", StallOffset64, StallOffset32},
    {"SetWatchdogTimer", SetWatchdogTimerOffset64, SetWatchdogTimerOffset32},
    {"ConnectController", ConnectControllerOffset64, ConnectControllerOffset32},
    {"DisconnectController", DisconnectControllerOffset64,
DisconnectControllerOffset32},
    {"OpenProtocol", OpenProtocolOffset64, OpenProtocolOffset32},
    {"CloseProtocol", CloseProtocolOffset64, CloseProtocolOffset32},
    {"OpenProtocolInformation", OpenProtocolInformationOffset64,
     OpenProtocolInformationOffset32},
    {"ProtocolsPerHandle", ProtocolsPerHandleOffset64, ProtocolsPerHandleOffset32},
    {"LocateHandleBuffer", LocateHandleBufferOffset64, LocateHandleBufferOffset32},
    {"LocateProtocol", LocateProtocolOffset64, LocateProtocolOffset32},
    {"InstallMultipleProtocolInterfaces", InstallMultipleProtocolInterfacesOffset64,
     InstallMultipleProtocolInterfacesOffset32},
    {"UninstallMultipleProtocolInterfaces",
UninstallMultipleProtocolInterfacesOffset64,
     UninstallMultipleProtocolInterfacesOffset32},
    {"CalculateCrc32", CalculateCrc32Offset64, CalculateCrc32Offset32},
    {"CopyMem", CopyMemOffset64, CopyMemOffset32},
    {"SetMem", SetMemOffset64, SetMemOffset32},
    {"CreateEventEx", CreateEventExOffset64, CreateEventExOffset32}};
size_t bootServicesTableAllLength = sizeof(bootServicesTableAll) / sizeof(service);

struct service runtimeServicesTableAll[] = {
    {"GetTime", GetTimeOffset64, GetTimeOffset32},
    {"SetTime", SetTimeOffset64, SetTimeOffset32},
    {"GetWakeupTime", GetWakeupTimeOffset64, GetWakeupTimeOffset32},
    {"SetWakeupTime", SetWakeupTimeOffset64, SetWakeupTimeOffset32},
    {"SetVirtualAddressMap", SetVirtualAddressMapOffset64,
SetVirtualAddressMapOffset32},
    {"ConvertPointer", ConvertPointerOffset64, ConvertPointerOffset32},
    {"GetVariable", GetVariableOffset64, GetVariableOffset32},
    {"GetNextVariableName", GetNextVariableNameOffset64, GetNextVariableNameOffset32},
    {"SetVariable", SetVariableOffset64, SetVariableOffset32},
    {"GetNextHighMonotonicCount", GetNextHighMonotonicCountOffset64,
     GetNextHighMonotonicCountOffset32},
    {"ResetSystem", ResetSystemOffset64, ResetSystemOffset32},
    {"UpdateCapsule", UpdateCapsuleOffset64, UpdateCapsuleOffset32},
    {"QueryCapsuleCapabilities", QueryCapsuleCapabilitiesOffset64,
     QueryCapsuleCapabilitiesOffset32},
    {"QueryVariableInfo", QueryVariableInfoOffset64, QueryVariableInfoOffset32}};
size_t runtimeServicesTableAllLength = sizeof(runtimeServicesTableAll)
/ sizeof(service);

struct service_info_64bit smmServicesProt64[] = {
    {"SmmInstallProtocolInterface", SmmInstallProtocolInterfaceOffset64, REG_RDX},
    {"SmmUninstallProtocolInterface", SmmUninstallProtocolInterfaceOffset64, REG_RDX},
    {"SmmHandleProtocol", SmmHandleProtocolOffset64, REG_RDX},
    {"SmmRegisterProtocolNotify", SmmRegisterProtocolNotifyOffset64, REG_RCX},
    {"SmmLocateHandle", SmmLocateHandleOffset64, REG_RDX},
    {"SmmLocateProtocol", SmmLocateProtocolOffset64, REG_RCX}};
size_t smmServicesProt64Length = sizeof(smmServicesProt64)
```

```
/ sizeof(service_info_64bit);

struct service smmServicesTableAll[] = {
    {"SmmInstallConfigurationTable", SmmInstallConfigurationTableOffset64,
     SmmInstallConfigurationTableOffset32},
    {"SmmAllocatePool", SmmAllocatePoolOffset64, SmmAllocatePoolOffset32},
    {"SmmFreePool", SmmFreePoolOffset64, SmmFreePoolOffset32},
    {"SmmAllocatePages", SmmAllocatePagesOffset64, SmmAllocatePagesOffset32},
    {"SmmFreePages", SmmFreePagesOffset64, SmmFreePagesOffset32},
    {"SmmStartupThisAp", SmmStartupThisApOffset64, SmmStartupThisApOffset32},
    {"SmmInstallProtocolInterface", SmmInstallProtocolInterfaceOffset64,
     SmmInstallProtocolInterfaceOffset32},
    {"SmmUninstallProtocolInterface", SmmUninstallProtocolInterfaceOffset64,
     SmmUninstallProtocolInterfaceOffset32},
    {"SmmHandleProtocol", SmmHandleProtocolOffset64, SmmHandleProtocolOffset32},
    {"SmmRegisterProtocolNotify", SmmRegisterProtocolNotifyOffset64,
     SmmRegisterProtocolNotifyOffset32},
    {"SmmLocateHandle", SmmLocateHandleOffset64, SmmLocateHandleOffset32},
    {"SmmLocateProtocol", SmmLocateProtocolOffset64, SmmLocateProtocolOffset32},
    {"SmiManage", SmiManageOffset64, SmiManageOffset32},
    {"SmiHandlerRegister", SmiHandlerRegisterOffset64, SmiHandlerRegisterOffset32},
    {"SmiHandlerUnRegister", SmiHandlerUnRegisterOffset64,
SmiHandlerUnRegisterOffset32}};
size_t smmServicesTableAllLength = sizeof(smmServicesTableAll) / sizeof(service);
}
```

efiXplorer/tables/efi_system_tables.h

```
/*
 * ____________________.___
 * \_   _____/\_   _____/|   |
 *  |    __)_  |    __)  |   |
 *  |        \ |     \   |   |
 * /_______  / \___  /   |___|
 *         \/      \/
 *   _________       .__                 ____  __.      .__  _____
 *  /   _____/_  _  _|__| ______ _____  |    |/ _| ____ |__|/ ____\____
 *  \_____  \\ \/ \/ /  |/  ___//  ___/ |      <  /    \|  \   __\/ __ \
 *  /        \\     /|  |\___ \ \___ \  |    |  \|   |  \  ||  | \  ___/
 * /_______  / \/\_/ |__/____  >____  > |____|__ \___|  /__||__|  \___  >
 *         \/                \/     \/          \/    \/              \/
 *
 * EFI Swiss Knife
 * An IDA plugin to improve (U)EFI reversing
 *
 * Copyright (C) 2016, 2017  Pedro Vilaça (fG!) - reverser@put.as -
 * https://reverse.put.as
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

```
 *
 * efi_system_tables.h
 *
 */

#pragma once

#include "stdint.h"

#define BTABLE_LEN 43
#define RTABLE_LEN 14

struct services_entry {
    char name[256];
    uint32_t offset64;
    uint32_t offset32;
    char description[1024];
    uint32_t nr_args;
    char prototype[512];
    char parameters[2048];
    char rcx_param[256];
    char rdx_param[256];
    char r8_param[256];
    char r9_param[256];
    char stack1_param[256];
    char stack2_param[256];
    char stack3_param[256];
    char stack4_param[256];
    uint32_t count;
};

struct services_entry boot_services_table[] = {
    {"RaiseTPL", 0x18, 0x18,
     "Raises a task's priority level and returns its previous level.", 1,
     "EFI_TPL(EFIAPI * EFI_RAISE_TPL) (IN EFI_TPL NewTpl)",
     "NewTpl   The new task priority level.", "IN EFI_TPL NewTpl", "", "", "",
"", "",
     "", 0},
    {"RestoreTPL", 0x20, 0x1c, "Restores a task's priority level to its
previous value.",
     1, "VOID(EFIAPI * EFI_RESTORE_TPL) (IN EFI_TPL OldTpl)",
     "OldTpl   The previous task priority level to restore.", "IN EFI_TPL OldTpl",
"", "",
     "", "", "", "", "", 0},
    {"AllocatePages", 0x28, 0x20, "Allocates memory pages from the system.", 4,
     "EFI_STATUS(EFIAPI * EFI_ALLOCATE_PAGES) (IN EFI_ALLOCATE_TYPE Type, IN "
     "EFI_MEMORY_TYPE MemoryType, IN UINTN Pages, IN OUT EFI_PHYSICAL_ADDRESS "
     "*Memory)",
     "Type         The type of allocation to perform.\n\
MemoryType   The type of memory to allocate.\n\
Pages        The number of contiguous 4 KB pages to allocate.\n\
Memory       The pointer to a physical address. On input, the way in which the address
is used depends on the value of Type.",
     "IN EFI_ALLOCATE_TYPE Type", "IN EFI_MEMORY_TYPE MemoryType", "IN UINTN Pages",
     "IN OUT EFI_PHYSICAL_ADDRESS *Memory", "", "", "", "", 0},
    {"FreePages", 0x30, 0x24, "Frees memory pages.", 2,
     "EFI_STATUS(EFIAPI * EFI_FREE_PAGES) (IN EFI_PHYSICAL_ADDRESS Memory, IN "
     "UINTN Pages)",
     "Memory   The base physical address of the pages to be freed.\n\
Pages    The number of contiguous 4 KB pages to free.",
     "IN EFI_PHYSICAL_ADDRESS Memory", "IN UINTN Pages", "", "", "", "", "", "", 0},
    {"GetMemoryMap", 0x38, 0x28, "Returns the current memory map.", 5,
     "EFI_STATUS(EFIAPI * EFI_GET_MEMORY_MAP) (IN OUT UINTN *MemoryMapSize, IN "
     "OUT EFI_MEMORY_DESCRIPTOR *MemoryMap, OUT UINTN *MapKey, OUT UINTN "
```

```
     "*DescriptorSize, OUT UINT32 *DescriptorVersion)",
     "MemoryMapSize       A pointer to the size, in bytes, of the MemoryMap buffer. On
input, this is the size of the buffer allocated by the caller.\n\
                    On output, it is the size of the buffer returned by the firmware if
the buffer was large enough, or the size of the buffer\n\
                    needed to contain the map if the buffer was too small.\n\
MemoryMap           A pointer to the buffer in which firmware places the current
memory map.\n\
MapKey              A pointer to the location in which firmware returns the key for the
current memory map.\n\
DescriptorSize      A pointer to the location in which firmware returns the size, in
bytes, of an individual EFI_MEMORY_DESCRIPTOR.\n\
DescriptorVersion   A pointer to the location in which firmware returns the version
number associated with the EFI_MEMORY_DESCRIPTOR.",
     "IN OUT UINTN *MemoryMapSize", "IN OUT EFI_MEMORY_DESCRIPTOR *MemoryMap",
     "OUT UINTN *MapKey", "OUT UINTN *DescriptorSize", "OUT UINT32 *DescriptorVersion",
     "", "", "", 0},
    {"AllocatePool", 0x40, 0x2c, "Allocates pool memory.", 3,
     "EFI_STATUS(EFIAPI * EFI_ALLOCATE_POOL) (IN EFI_MEMORY_TYPE PoolType, IN "
     "UINTN Size, OUT VOID **Buffer)",
     "PoolType   The type of pool to allocate.\n\
Size       The number of bytes to allocate from the pool.\n\
Buffer     A pointer to a pointer to the allocated buffer if the call succeeds;
undefined otherwise.",
     "IN EFI_MEMORY_TYPE PoolType", "IN UINTN Size", "OUT VOID **Buffer", "", "",
"", "",
     "", 0},
    {"FreePool", 0x48, 0x30, "Returns pool memory to the system.", 1,
     "EFI_STATUS(EFIAPI * EFI_FREE_POOL) (IN VOID *Buffer)",
     "Buffer The pointer to the buffer to free.", "IN VOID *Buffer", "", "", "",
"", "",
     "", "", 0},
    {"CreateEvent", 0x50, 0x34, "Creates an event.", 5,
     "EFI_STATUS(EFIAPI * EFI_CREATE_EVENT) (IN UINT32 Type, IN EFI_TPL "
     "NotifyTpl, IN EFI_EVENT_NOTIFY NotifyFunction, IN VOID *NotifyContext, "
     "OUT EFI_EVENT *Event)",
     "Type             The type of event to create and its mode and attributes.\n\
NotifyTpl        The task priority level of event notifications, if needed.\n\
NotifyFunction   The pointer to the event's notification function, if any.\n\
NotifyContext    The pointer to the notification function's context; corresponds to
parameter Context in the notification function.\n\
Event            The pointer to the newly created event if the call succeeds;
undefined otherwise.",
     "IN UINT32 Type", "IN EFI_TPL NotifyTpl", "IN EFI_EVENT_NOTIFY NotifyFunction",
     "IN VOID *NotifyContext", "OUT EFI_EVENT *Event", "", "", "", 0},
    {"SetTimer", 0x58, 0x38,
     "Sets the type of timer and the trigger time for a timer event.", 3,
     "EFI_STATUS(EFIAPI * EFI_SET_TIMER) (IN EFI_EVENT Event, IN "
     "EFI_TIMER_DELAY Type, IN UINT64 TriggerTime)",
     "Event         The timer event that is to be signaled at the specified time.\n\
Type          The type of time that is specified in TriggerTime.\n\
TriggerTime   The number of 100ns units until the timer expires. A TriggerTime of 0 is
legal. If Type is TimerRelative and TriggerTime is 0, then the timer event will be
signaled on the next timer tick. If Type is TimerPeriodic and TriggerTime is 0, then
the timer event will be signaled on every timer tick.",
     "IN EFI_EVENT Event", "IN EFI_TIMER_DELAY Type", "IN UINT64 TriggerTime", "",
"", "",
     "", "", 0},
    {"WaitForEvent", 0x60, 0x3c, "Stops execution until an event is signaled.", 3,
     "EFI_STATUS(EFIAPI * EFI_WAIT_FOR_EVENT) (IN UINTN NumberOfEvents, IN "
     "EFI_EVENT *Event, OUT UINTN *Index)",
     "NumberOfEvents   The number of events in the Event array.\n\
Event            An array of EFI_EVENT.\n\
Index            The pointer to the index of the event which satisfied the
```

```
wait condition.",
     "IN UINTN NumberOfEvents", "IN EFI_EVENT *Event", "OUT UINTN *Index", "", "",
"", "",
     "", 0},
    {"SignalEvent", 0x68, 0x40, "Signals an event.", 1,
     "EFI_STATUS(EFIAPI * EFI_SIGNAL_EVENT) (IN EFI_EVENT Event)",
     "Event The event to signal.", "IN EFI_EVENT Event", "", "", "", "", "", "",
"", 0},
    {"CloseEvent", 0x70, 0x44, "Closes an event.", 1,
     "EFI_STATUS(EFIAPI * EFI_CLOSE_EVENT) (IN EFI_EVENT Event)",
     "Event The event to close.", "IN EFI_EVENT Event", "", "", "", "", "", "", "", 0},
    {"CheckEvent", 0x78, 0x48, "Checks whether an event is in the signaled state.", 1,
     "EFI_STATUS(EFIAPI * EFI_CHECK_EVENT) (IN EFI_EVENT Event)",
     "Event The event to check.", "IN EFI_EVENT Event", "", "", "", "", "", "", "", 0},
    {"InstallProtocolInterface", 0x80, 0x4c,
     "Installs a protocol interface on a device handle. If the handle does not "
     "exist, it is created and added to the list of handles in the system. "
     "InstallMultipleProtocolInterfaces() performs more error checking than "
     "InstallProtocolInterface(), so it is recommended that "
     "InstallMultipleProtocolInterfaces() be used in place of "
     "InstallProtocolInterface()",
     4,
     "EFI_STATUS(EFIAPI * EFI_INSTALL_PROTOCOL_INTERFACE) (IN OUT EFI_HANDLE "
     "*Handle, IN EFI_GUID *Protocol, IN EFI_INTERFACE_TYPE InterfaceType, IN "
     "VOID *Interface)",
     "Handle          A pointer to the EFI_HANDLE on which the interface is to
be installed.\n\
Protocol        The numeric ID of the protocol interface.\n\
InterfaceType   Indicates whether Interface is supplied in native form.\n\
Interface       A pointer to the protocol interface.",
     "IN OUT EFI_HANDLE *Handle", "IN EFI_GUID *Protocol",
     "IN EFI_INTERFACE_TYPE InterfaceType", "IN VOID *Interface", "", "", "", "", 0},
    {"ReinstallProtocolInterface", 0x88, 0x50,
     "Reinstalls a protocol interface on a device handle.", 4,
     "EFI_STATUS(EFIAPI * EFI_REINSTALL_PROTOCOL_INTERFACE) (IN EFI_HANDLE "
     "Handle, IN EFI_GUID *Protocol, IN VOID *OldInterface, IN VOID "
     "*NewInterface)",
     "Handle         Handle on which the interface is to be reinstalled.\n\
Protocol       The numeric ID of the interface.\n\
OldInterface   A pointer to the old interface. NULL can be used if a structure is not
associated with Protocol.\n\
NewInterface   A pointer to the new interface.",
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol", "IN VOID *OldInterface",
     "IN VOID *NewInterface", "", "", "", "", 0},
    {"UninstallProtocolInterface", 0x90, 0x54,
     "Removes a protocol interface from a device handle. It is recommended "
     "that UninstallMultipleProtocolInterfaces() be used in place of "
     "UninstallProtocolInterface().",
     3,
     "EFI_STATUS(EFIAPI * EFI_UNINSTALL_PROTOCOL_INTERFACE) (IN EFI_HANDLE "
     "Handle, IN EFI_GUID *Protocol, IN VOID *Interface)",
     "Handle      The handle on which the interface was installed.\n\
Protocol    The numeric ID of the interface.\n\
Interface   A pointer to the interface.",
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol", "IN VOID *Interface", "", "", "",
     "", "", 0},
    {"HandleProtocol", 0x98, 0x58,
     "Queries a handle to determine if it supports a specified protocol.", 3,
     "EFI_STATUS(EFIAPI * EFI_HANDLE_PROTOCOL) (IN EFI_HANDLE Handle, IN "
     "EFI_GUID *Protocol, OUT VOID **Interface)",
     "Handle      The handle being queried.\n\
Protocol    The published unique identifier of the protocol.\n\
Interface   Supplies the address where a pointer to the corresponding Protocol
Interface is returned.",
```

```
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol", "OUT VOID **Interface", "",
"", "",
     "", "", 0},
    {"RegisterProtocolNotify", 0xA8, 0x60,
     "Creates an event that is to be signaled whenever an interface is "
     "installed for a specified protocol.",
     3,
     "EFI_STATUS(EFIAPI * EFI_REGISTER_PROTOCOL_NOTIFY) (IN EFI_GUID "
     "*Protocol, IN EFI_EVENT Event, OUT VOID **Registration)",
     "Protocol       The numeric ID of the protocol for which the event is to
be registered.\n\
Event          Event that is to be signaled whenever a protocol interface is registered
for Protocol.\n\
Registration   A pointer to a memory location to receive the registration value.",
     "IN EFI_GUID *Protocol", "IN EFI_EVENT Event", "OUT VOID **Registration", "",
"", "",
     "", "", 0},
    {"LocateHandle", 0xB0, 0x64,
     "Returns an array of handles that support a specified protocol.", 5,
     "EFI_STATUS(EFIAPI * EFI_LOCATE_HANDLE) (IN EFI_LOCATE_SEARCH_TYPE "
     "SearchType, IN EFI_GUID *Protocol, OPTIONAL IN VOID *SearchKey, OPTIONAL "
     "IN OUT UINTN *BufferSize, OUT EFI_HANDLE *Buffer)",
     "SearchType   Specifies which handle(s) are to be returned.\n\
Protocol     Specifies the protocol to search by.\n\
SearchKey    Specifies the search key.\n\
BufferSize   On input, the size in bytes of Buffer. On output, the size in bytes of the
array returned in Buffer (if the buffer was large enough) or the size, in bytes, of the
buffer needed to obtain the array (if the buffer was not large enough).\n\
Buffer       The buffer in which the array is returned.",
     "IN EFI_LOCATE_SEARCH_TYPE SearchType", "IN EFI_GUID *Protocol",
     "OPTIONAL IN VOID *SearchKey", "OPTIONAL IN OUT UINTN *BufferSize",
     "OUT EFI_HANDLE *Buffer", "", "", "", 0},
    {"LocateDevicePath", 0xB8, 0x68,
     "Locates the handle to a device on the device path that supports the "
     "specified protocol.",
     3,
     "EFI_STATUS(EFIAPI * EFI_LOCATE_DEVICE_PATH) (IN EFI_GUID *Protocol, IN "
     "OUT EFI_DEVICE_PATH_PROTOCOL **DevicePath, OUT EFI_HANDLE *Device)",
     "Protocol     Specifies the protocol to search for.\n\
DevicePath   On input, a pointer to a pointer to the device path. On output, the device
path pointer is modified to point to the remaining part of the device path.\n\
Device       A pointer to the returned device handle.",
     "IN EFI_GUID *Protocol", "IN OUT EFI_DEVICE_PATH_PROTOCOL **DevicePath",
     "OUT EFI_HANDLE *Device", "", "", "", "", "", 0},
    {"InstallConfigurationTable", 0xC0, 0x6c,
     "Adds, updates, or removes a configuration table entry from the EFI "
     "System Table.",
     2,
     "EFI_STATUS(EFIAPI * EFI_INSTALL_CONFIGURATION_TABLE) (IN EFI_GUID *Guid, "
     "IN VOID *Table)",
     "Guid    A pointer to the GUID for the entry to add, update, or remove.\n\
Table   A pointer to the configuration table for the entry to add, update, or remove.
May be NULL.",
     "IN EFI_GUID *Guid", "IN VOID *Table", "", "", "", "", "", "", 0},
    {"LoadImage", 0xC8, 0x70, "Loads an EFI image into memory.", 6,
     "EFI_STATUS(EFIAPI * EFI_IMAGE_LOAD) (IN BOOLEAN BootPolicy, IN "
     "EFI_HANDLE ParentImageHandle, IN EFI_DEVICE_PATH_PROTOCOL *DevicePath, "
     "IN VOID *SourceBuffer OPTIONAL, IN UINTN SourceSize, OUT EFI_HANDLE "
     "*ImageHandle)",
     "BootPolicy          If TRUE, indicates that the request originates from the boot
manager, and that the boot manager is attempting to load FilePath as a boot selection.
Ignored if SourceBuffer is not NULL.\n\
ParentImageHandle   The caller's image handle.\n\
DevicePath          The DeviceHandle specific file path from which the image
```

```
is loaded.\n\
SourceBuffer        If not NULL, a pointer to the memory location containing a copy of
the image to be loaded.\n\
SourceSize          The size in bytes of SourceBuffer. Ignored if SourceBuffer
is NULL.\n\
ImageHandle         The pointer to the returned image handle that is created when the
image is successfully loaded.",
     "IN BOOLEAN BootPolicy", "IN EFI_HANDLE ParentImageHandle",
     "IN EFI_DEVICE_PATH_PROTOCOL *DevicePath", "IN VOID *SourceBuffer OPTIONAL",
     "IN UINTN SourceSize", "OUT EFI_HANDLE *ImageHandle", "", "", 0},
    {"StartImage", 0xD0, 0x74, "Transfers control to a loaded image's entry point.", 3,
     "EFI_STATUS(EFIAPI * EFI_IMAGE_START) (IN EFI_HANDLE ImageHandle, OUT "
     "UINTN *ExitDataSize, OUT CHAR16 **ExitData OPTIONAL)",
     "ImageHandle    Handle of image to be started.\n\
ExitDataSize   The pointer to the size, in bytes, of ExitData.\n\
ExitData       The pointer to a pointer to a data buffer that includes a Null-
terminated string, optionally followed by additional binary data.",
     "IN EFI_HANDLE ImageHandle", "OUT UINTN *ExitDataSize",
     "OUT CHAR16 **ExitData OPTIONAL", "", "", "", "", "", 0},
    {"Exit", 0xD8, 0x78,
     "Terminates a loaded EFI image and returns control to boot services.", 4,
     "EFI_STATUS(EFIAPI * EFI_EXIT) (IN EFI_HANDLE ImageHandle, IN EFI_STATUS "
     "ExitStatus, IN UINTN ExitDataSize, IN CHAR16 *ExitData OPTIONAL)",
     "ImageHandle    Handle that identifies the image. This parameter is passed to the
image on entry.\n\
ExitStatus     The image's exit code.\n\
ExitDataSize   The size, in bytes, of ExitData. Ignored if ExitStatus
is EFI_SUCCESS.\n\
ExitData       The pointer to a data buffer that includes a Null-terminated string,
optionally followed by additional binary data. The string is a description that the
caller may use to further indicate the reason for the image's exit. ExitData is only
valid if ExitStatus is something other than EFI_SUCCESS. The ExitData buffer must be
allocated by calling AllocatePool().",
     "IN EFI_HANDLE ImageHandle", "IN EFI_STATUS ExitStatus", "IN UINTN ExitDataSize",
     "IN CHAR16 *ExitData OPTIONAL", "", "", "", "", 0},
    {"UnloadImage", 0xE0, 0x7c, "Unloads an image.", 1,
     "EFI_STATUS(EFIAPI * EFI_IMAGE_UNLOAD) (IN EFI_HANDLE ImageHandle)",
     "ImageHandle Handle that identifies the image to be unloaded.",
     "IN EFI_HANDLE ImageHandle", "", "", "", "", "", "", "", 0},
    {"ExitBootServices", 0xE8, 0x80, "Terminates all boot services.", 2,
     "EFI_STATUS(EFIAPI * EFI_EXIT_BOOT_SERVICES) (IN EFI_HANDLE ImageHandle, "
     "IN UINTN MapKey)",
     "ImageHandle   Handle that identifies the exiting image.\n\
MapKey         Key to the latest memory map.",
     "IN EFI_HANDLE ImageHandle", "IN UINTN MapKey", "", "", "", "", "", "", 0},
    {"GetNextMonotonicCount", 0xF0, 0x84,
     "Returns a monotonically increasing count for the platform.", 1,
     "EFI_STATUS(EFIAPI * EFI_GET_NEXT_MONOTONIC_COUNT) (OUT UINT64 *Count)",
     "Count The pointer to returned value.", "OUT UINT64 *Count", "", "", "", "",
"", "",
     "", 0},
    {"Stall", 0xF8, 0x88, "Induces a fine-grained stall.", 1,
     "EFI_STATUS(EFIAPI * EFI_STALL) (IN UINTN Microseconds)",
     "Microseconds The number of microseconds to stall execution.",
     "IN UINTN Microseconds", "", "", "", "", "", "", "", 0},
    {"SetWatchdogTimer", 0x100, 0x8c, "Sets the system's watchdog timer.", 4,
     "EFI_STATUS(EFIAPI * EFI_SET_WATCHDOG_TIMER) (IN UINTN Timeout, IN UINT64 "
     "WatchdogCode, IN UINTN DataSize, IN CHAR16 *WatchdogData OPTIONAL)",
     "Timeout        The number of seconds to set the watchdog timer to.\n\
WatchdogCode   The numeric code to log on a watchdog timer timeout event.\n\
DataSize       The size, in bytes, of WatchdogData.\n\
WatchdogData   A data buffer that includes a Null-terminated string, optionally
followed by additional binary data.",
     "IN UINTN Timeout", "IN UINT64 WatchdogCode", "IN UINTN DataSize",
```

```
     "IN CHAR16 *WatchdogData OPTIONAL", "", "", "", "", 0},
    {"ConnectController", 0x108, 0x90, "Connects one or more drivers to a
controller.", 4,
     "EFI_STATUS(EFIAPI * EFI_CONNECT_CONTROLLER) (IN EFI_HANDLE "
     "ControllerHandle, IN EFI_HANDLE *DriverImageHandle, OPTIONAL IN "
     "EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, OPTIONAL IN BOOLEAN "
     "Recursive)",
     "ControllerHandle      The handle of the controller to which driver(s) are to
be connected.\n\
DriverImageHandle     A pointer to an ordered list handles that support the
EFI_DRIVER_BINDING_PROTOCOL.\n\
RemainingDevicePath   A pointer to the device path that specifies a child of the
controller specified by ControllerHandle.\n\
Recursive             If TRUE, then ConnectController() is called recursively until the
entire tree of controllers below the controller specified by ControllerHandle have been
created. If FALSE, then the tree of controllers is only expanded one level.",
     "IN EFI_HANDLE ControllerHandle", "IN EFI_HANDLE *DriverImageHandle",
     "OPTIONAL IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath",
     "OPTIONAL IN BOOLEAN Recursive", "", "", "", "", 0},
    {"DisconnectController", 0x110, 0x94,
     "Disconnects one or more drivers from a controller.", 3,
     "EFI_STATUS(EFIAPI * EFI_DISCONNECT_CONTROLLER) (IN EFI_HANDLE "
     "ControllerHandle, IN EFI_HANDLE DriverImageHandle, OPTIONAL IN "
     "EFI_HANDLE ChildHandle OPTIONAL)",
     "ControllerHandle    The handle of the controller from which driver(s) are to
be disconnected.\n\
DriverImageHandle   The driver to disconnect from ControllerHandle. If
DriverImageHandle is NULL, then all the drivers currently managing ControllerHandle are
disconnected from ControllerHandle.\n\
ChildHandle         The handle of the child to destroy. If ChildHandle is NULL, then
all the children of ControllerHandle are destroyed before the drivers are disconnected
from ControllerHandle.",
     "IN EFI_HANDLE ControllerHandle", "IN EFI_HANDLE DriverImageHandle",
     "OPTIONAL IN EFI_HANDLE ChildHandle OPTIONAL", "", "", "", "", "", 0},
    {"OpenProtocol", 0x118, 0x98,
     "Queries a handle to determine if it supports a specified protocol. If "
     "the protocol is supported by the handle, it opens the protocol on behalf "
     "of the calling agent.",
     6,
     "EFI_STATUS(EFIAPI * EFI_OPEN_PROTOCOL) (IN EFI_HANDLE Handle, IN "
     "EFI_GUID *Protocol, OUT VOID **Interface, OPTIONAL IN EFI_HANDLE "
     "AgentHandle, IN EFI_HANDLE ControllerHandle, IN UINT32 Attributes)",
     "Handle             The handle for the protocol interface that is being opened.\n\
Protocol           The published unique identifier of the protocol.\n\
Interface          Supplies the address where a pointer to the corresponding Protocol
Interface is returned.\n\
AgentHandle        The handle of the agent that is opening the protocol interface
specified by Protocol and Interface.\n\
ControllerHandle   If the agent that is opening a protocol is a driver that follows the
UEFI Driver Model, then this parameter is the controller handle that requires the
protocol interface. If the agent does not follow the UEFI Driver Model, then this
parameter is optional and may be NULL.\n\
Attributes         The open mode of the protocol interface specified by Handle
and Protocol.",
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol", "OUT VOID **Interface",
     "OPTIONAL IN EFI_HANDLE AgentHandle", "IN EFI_HANDLE ControllerHandle",
     "IN UINT32 Attributes", "", "", 0},
    {"CloseProtocol", 0x120, 0x9c,
     "Closes a protocol on a handle that was opened using OpenProtocol().", 4,
     "EFI_STATUS(EFIAPI * EFI_CLOSE_PROTOCOL) (IN EFI_HANDLE Handle, IN "
     "EFI_GUID *Protocol, IN EFI_HANDLE AgentHandle, IN EFI_HANDLE "
     "ControllerHandle)",
     "Handle             The handle for the protocol interface that was previously
opened with OpenProtocol(), and is now being closed.\n\
```

```
Protocol           The published unique identifier of the protocol.\n\
AgentHandle        The handle of the agent that is closing the protocol interface.\n\
ControllerHandle   If the agent that opened a protocol is a driver that follows the
UEFI Driver Model, then this parameter is the controller handle that required the
protocol interface.",
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol", "IN EFI_HANDLE AgentHandle",
     "IN EFI_HANDLE ControllerHandle", "", "", "", "", 0},
    {"OpenProtocolInformation", 0x128, 0xa0,
     "Retrieves the list of agents that currently have a protocol interface "
     "opened.",
     4,
     "EFI_STATUS(EFIAPI * EFI_OPEN_PROTOCOL_INFORMATION) (IN EFI_HANDLE "
     "Handle, IN EFI_GUID *Protocol, OUT EFI_OPEN_PROTOCOL_INFORMATION_ENTRY "
     "**EntryBuffer, OUT UINTN *EntryCount)",
     "Handle        The handle for the protocol interface that is being queried.\n\
Protocol      The published unique identifier of the protocol.\n\
EntryBuffer   A pointer to a buffer of open protocol information in the form of
EFI_OPEN_PROTOCOL_INFORMATION_ENTRY structures.\n\
EntryCount    A pointer to the number of entries in EntryBuffer.",
     "IN EFI_HANDLE Handle", "IN EFI_GUID *Protocol",
     "OUT EFI_OPEN_PROTOCOL_INFORMATION_ENTRY **EntryBuffer", "OUT UINTN
*EntryCount", "",
     "", "", "", 0},
    {"ProtocolsPerHandle", 0x130, 0xa4,
     "Retrieves the list of protocol interface GUIDs that are installed on a "
     "handle in a buffer allocated from pool.",
     3,
     "EFI_STATUS(EFIAPI * EFI_PROTOCOLS_PER_HANDLE) (IN EFI_HANDLE Handle, OUT "
     "EFI_GUID ***ProtocolBuffer, OUT UINTN *ProtocolBufferCount)",
     "Handle                The handle from which to retrieve the list of protocol
interface GUIDs.\n\
ProtocolBuffer        A pointer to the list of protocol interface GUID pointers that
are installed on Handle.\n\
ProtocolBufferCount   A pointer to the number of GUID pointers present
in ProtocolBuffer.",
     "IN EFI_HANDLE Handle", "OUT EFI_GUID ***ProtocolBuffer",
     "OUT UINTN *ProtocolBufferCount", "", "", "", "", "", 0},
    {"LocateHandleBuffer", 0x138, 0xa8,
     "Returns an array of handles that support the requested protocol in a "
     "buffer allocated from pool.",
     5,
     "EFI_STATUS(EFIAPI * EFI_LOCATE_HANDLE_BUFFER) (IN EFI_LOCATE_SEARCH_TYPE "
     "SearchType, IN EFI_GUID *Protocol, OPTIONAL IN VOID *SearchKey, OPTIONAL "
     "IN OUT UINTN *NoHandles, OUT EFI_HANDLE **Buffer)",
     "SearchType   Specifies which handle(s) are to be returned.\n\
Protocol     Provides the protocol to search by. This parameter is only valid for a
SearchType of ByProtocol.\n\
SearchKey    Supplies the search key depending on the SearchType.\n\
NoHandles    The number of handles returned in Buffer.\n\
Buffer       A pointer to the buffer to return the requested array of handles that
support Protocol.",
     "IN EFI_LOCATE_SEARCH_TYPE SearchType", "IN EFI_GUID *Protocol",
     "OPTIONAL IN VOID *SearchKey", "OPTIONAL IN OUT UINTN *NoHandles",
     "OUT EFI_HANDLE **Buffer", "", "", "", 0},
    {"LocateProtocol", 0x140, 0xac,
     "Returns the first protocol instance that matches the given protocol.", 3,
     "EFI_STATUS(EFIAPI * EFI_LOCATE_PROTOCOL) (IN EFI_GUID *Protocol, IN VOID "
     "*Registration, OPTIONAL OUT VOID **Interface)",
     "Protocol       Provides the protocol to search for.\n\
Registration   Optional registration key returned from RegisterProtocolNotify().\n\
Interface      On return, a pointer to the first interface that matches Protocol
and Registration.",
     "IN EFI_GUID *Protocol", "IN VOID *Registration", "OPTIONAL OUT VOID **Interface",
     "", "", "", "", "", 0},
```

```
    {"InstallMultipleProtocolInterfaces", 0x148, 0xb0,
     "Installs one or more protocol interfaces into the boot services "
     "environment.",
     1,
     "EFI_STATUS(EFIAPI * EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES) (IN OUT "
     "EFI_HANDLE *Handle,...)",
     "Handle   The pointer to a handle to install the new protocol interfaces on, or a
pointer to NULL if a new handle is to be allocated.\n\
...      A variable argument list containing pairs of protocol GUIDs and
protocol interfaces.",
     "IN OUT EFI_HANDLE *Handle", "", "", "", "", "", "", "", 0},
    {"UninstallMultipleProtocolInterfaces", 0x150, 0xb4,
     "Removes one or more protocol interfaces into the boot services "
     "environment.",
     1,
     "EFI_STATUS(EFIAPI * EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES) (IN "
     "EFI_HANDLE Handle,...)",
     "Handle   The handle to remove the protocol interfaces from.\n\
...      A variable argument list containing pairs of protocol GUIDs and
protocol interfaces.",
     "IN EFI_HANDLE Handle", "", "", "", "", "", "", "", 0},
    {"CalculateCrc32", 0x158, 0xb8,
     "Computes and returns a 32-bit CRC for a data buffer.", 3,
     "EFI_STATUS(EFIAPI * EFI_CALCULATE_CRC32) (IN VOID *Data, IN UINTN "
     "DataSize, OUT UINT32 *Crc32)",
     "Data       A pointer to the buffer on which the 32-bit CRC is to be computed.\n\
DataSize   The number of bytes in the buffer Data.\n\
Crc32      The 32-bit CRC that was computed for the data buffer specified by Data
and DataSize.",
     "IN VOID *Data", "IN UINTN DataSize", "OUT UINT32 *Crc32", "", "", "", "", "", 0},
    {"CopyMem", 0x160, 0xbc, "Copies the contents of one buffer to another buffer.", 3,
     "VOID(EFIAPI * EFI_COPY_MEM) (IN VOID *Destination, IN VOID *Source, IN "
     "UINTN Length)",
     "Destination   The pointer to the destination buffer of the memory copy.\n\
Source        The pointer to the source buffer of the memory copy.\n\
Length        Number of bytes to copy from Source to Destination.",
     "IN VOID *Destination", "IN VOID *Source", "IN UINTN Length", "", "", "", "",
"", 0},
    {"SetMem", 0x168, 0xc0,
     "The SetMem() function fills a buffer with a specified value.", 3,
     "VOID(EFIAPI * EFI_SET_MEM) (IN VOID *Buffer, IN UINTN Size, IN UINT8 "
     "Value)",
     "Buffer   The pointer to the buffer to fill.\n\
Size     Number of bytes in Buffer to fill.\n\
Value    Value to fill Buffer with.",
     "IN VOID *Buffer", "IN UINTN Size", "IN UINT8 Value", "", "", "", "", "", 0},
    {"CreateEventEx", 0x170, 0xc4, "Creates an event in a group.", 6,
     "EFI_STATUS(EFIAPI * EFI_CREATE_EVENT_EX) (IN UINT32 Type, IN EFI_TPL "
     "NotifyTpl, IN EFI_EVENT_NOTIFY NotifyFunction OPTIONAL, IN CONST VOID "
     "*NotifyContext OPTIONAL, IN CONST EFI_GUID *EventGroup OPTIONAL, OUT "
     "EFI_EVENT *Event)",
     "Type             The type of event to create and its mode and attributes.\n\
NotifyTpl        The task priority level of event notifications,if needed.\n\
NotifyFunction   The pointer to the event's notification function, if any.\n\
NotifyContext    The pointer to the notification function's context; corresponds to
parameter Context in the notification function.\n\
EventGroup       The pointer to the unique identifier of the group to which this event
belongs. If this is NULL, then the function behaves as if the parameters were passed
to CreateEvent.\n\
Event            The pointer to the newly created event if the call succeeds;
undefined otherwise.",
     "IN UINT32 Type", "IN EFI_TPL NotifyTpl",
     "IN EFI_EVENT_NOTIFY NotifyFunction OPTIONAL",
     "IN CONST VOID *NotifyContext OPTIONAL", "IN CONST EFI_GUID *EventGroup OPTIONAL",
```

```
     "OUT EFI_EVENT *Event", "", "", 0}};

struct services_entry runtime_services_table[] = {
    {"GetTime", 0x18, 0x18,
     "Returns the current time and date information, and the time-keeping "
     "capabilities of the hardware platform.",
     2,
     "EFI_STATUS(EFIAPI * EFI_GET_TIME) (OUT EFI_TIME *Time, OUT "
     "EFI_TIME_CAPABILITIES *Capabilities OPTIONAL)",
     "Time           A pointer to storage to receive a snapshot of the current time.\n\
Capabilities   An optional pointer to a buffer to receive the real time clock
device's capabilities.",
     "OUT EFI_TIME *Time", "OUT EFI_TIME_CAPABILITIES *Capabilities OPTIONAL", "",
"", "",
     "", "", "", 0},
    {"SetTime", 0x20, 0x1c, "Sets the current local time and date information.", 1,
     "EFI_STATUS(EFIAPI * EFI_SET_TIME) (IN EFI_TIME *Time)",
     "Time A pointer to the current time.", "IN EFI_TIME *Time", "", "", "", "",
"", "",
     "", 0},
    {"GetWakeupTime", 0x28, 0x20, "Returns the current wakeup alarm clock setting.", 3,
     "EFI_STATUS(EFIAPI * EFI_GET_WAKEUP_TIME) (OUT BOOLEAN *Enabled, OUT "
     "BOOLEAN *Pending, OUT EFI_TIME *Time)",
     "Enabled   Indicates if the alarm is currently enabled or disabled.\n\
Pending   Indicates if the alarm signal is pending and requires acknowledgement.\n\
Time      The current alarm setting.",
     "OUT BOOLEAN *Enabled", "OUT BOOLEAN *Pending", "OUT EFI_TIME *Time", "", "",
"", "",
     "", 0},
    {"SetWakeupTime", 0x30, 0x24, "Sets the system wakeup alarm clock time.", 2,
     "EFI_STATUS(EFIAPI * EFI_SET_WAKEUP_TIME) (IN BOOLEAN Enable, IN EFI_TIME "
     "*Time OPTIONAL)",
     "Enabled   Enable or disable the wakeup alarm.\n\
Time      If Enable is TRUE, the time to set the wakeup alarm for. If Enable is FALSE,
then this parameter is optional, and may be NULL.",
     "IN BOOLEAN Enable", "IN EFI_TIME *Time OPTIONAL", "", "", "", "", "", "", 0},
    {"SetVirtualAddressMap", 0x38, 0x28,
     "Changes the runtime addressing mode of EFI firmware from physical to "
     "virtual.",
     4,
     "EFI_STATUS(EFIAPI * EFI_SET_VIRTUAL_ADDRESS_MAP) (IN UINTN "
     "MemoryMapSize, IN UINTN DescriptorSize, IN UINT32 DescriptorVersion, IN "
     "EFI_MEMORY_DESCRIPTOR *VirtualMap)",
     "MemoryMapSize       The size in bytes of VirtualMap.\n\
DescriptorSize      The size in bytes of an entry in the VirtualMap.\n\
DescriptorVersion   The version of the structure entries in VirtualMap.\n\
VirtualMap          An array of memory descriptors which contain new virtual address
mapping information for all runtime ranges.",
     "IN UINTN MemoryMapSize", "IN UINTN DescriptorSize", "IN
UINT32 DescriptorVersion",
     "IN EFI_MEMORY_DESCRIPTOR *VirtualMap", "", "", "", "", 0},
    {"ConvertPointer", 0x40, 0x2c,
     "Determines the new virtual address that is to be used on subsequent "
     "memory accesses.",
     2,
     "EFI_STATUS(EFIAPI * EFI_CONVERT_POINTER) (IN UINTN DebugDisposition, IN "
     "OUT VOID **Address)",
     "DebugDisposition   Supplies type information for the pointer being converted.\n\
Address            A pointer to a pointer that is to be fixed to be the value needed
for the new virtual address mappings being applied.",
     "IN UINTN DebugDisposition", "IN OUT VOID **Address", "", "", "", "", "", "", 0},
    {"GetVariable", 0x48, 0x30, "Returns the value of a variable.", 5,
     "EFI_STATUS(EFIAPI * EFI_GET_VARIABLE) (IN CHAR16 *VariableName, IN "
     "EFI_GUID *VendorGuid, OUT UINT32 *Attributes, OPTIONAL IN OUT UINTN "
```

```
     "*DataSize, OUT VOID *Data)",
     "VariableName   A Null-terminated string that is the name of the
vendor's variable.\n\
VendorGuid     A unique identifier for the vendor.\n\
Attributes     If not NULL, a pointer to the memory location to return the attributes
bitmask for the variable.\n\
DataSize       On input, the size in bytes of the return Data buffer. On output the
size of data returned in Data.\n\
Data           The buffer to return the contents of the variable.",
     "IN CHAR16 *VariableName", "IN EFI_GUID *VendorGuid", "OUT UINT32 *Attributes",
     "OPTIONAL IN OUT UINTN *DataSize", "OUT VOID *Data", "", "", "", 0},
    {"GetNextVariableName", 0x50, 0x3c, "Enumerates the current variable names.", 3,
     "EFI_STATUS(EFIAPI * EFI_GET_NEXT_VARIABLE_NAME) (IN OUT UINTN "
     "*VariableNameSize, IN OUT CHAR16 *VariableName, IN OUT EFI_GUID "
     "*VendorGuid)",
     "VariableNameSize   The size of the VariableName buffer.\n\
VariableName       On input, supplies the last VariableName that was returned
by GetNextVariableName(). On output, returns the Nullterminated string of the
current variable.\n\
VendorGuid         On input, supplies the last VendorGuid that was returned by
GetNextVariableName(). On output, returns the VendorGuid of the current variable.",
     "IN OUT UINTN *VariableNameSize", "IN OUT CHAR16 *VariableName",
     "IN OUT EFI_GUID *VendorGuid", "", "", "", "", "", 0},
    {"SetVariable", 0x58, 0x38, "Sets the value of a variable.", 5,
     "EFI_STATUS(EFIAPI * EFI_SET_VARIABLE) (IN CHAR16 *VariableName, IN "
     "EFI_GUID *VendorGuid, IN UINT32 Attributes, IN UINTN DataSize, IN VOID "
     "*Data)",
     "VariableName   A Null-terminated string that is the name of the vendor's
variable. Each VariableName is unique for each VendorGuid. VariableName must contain
1 or more characters. If VariableName is an empty string, then EFI_INVALID_PARAMETER
is returned.\n\
VendorGuid     A unique identifier for the vendor.\n\
Attributes     Attributes bitmask to set for the variable.\n\
DataSize       The size in bytes of the Data buffer. Unless the
EFI_VARIABLE_APPEND_WRITE, EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS, or
EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute is set, a size of zero
causes the variable to be deleted. When the EFI_VARIABLE_APPEND_WRITE attribute is set,
\nthen a SetVariable() call with a DataSize of zero will not cause any change to the
variable value (the timestamp associated with the variable may be updated however even
if no new data value is provided,\n see the description of the
EFI_VARIABLE_AUTHENTICATION_2 descriptor below. In this case the DataSize will not be
zero since the EFI_VARIABLE_AUTHENTICATION_2 descriptor will be populated).\n\
Data           The contents for the variable.",
     "IN CHAR16 *VariableName", "IN EFI_GUID *VendorGuid", "IN UINT32 Attributes",
     "IN UINTN DataSize", "IN VOID *Data", "", "", "", 0},
    {"GetNextHighMonotonicCount", 0x60, 0x3c,
     "Returns the next high 32 bits of the platform's monotonic counter.", 1,
     "EFI_STATUS(EFIAPI * EFI_GET_NEXT_HIGH_MONO_COUNT) (OUT UINT32 "
     "*HighCount)",
     "HighCount The pointer to returned value.", "OUT UINT32 *HighCount", "", "",
"", "",
     "", "", "", 0},
    {"ResetSystem", 0x68, 0x40, "Resets the entire platform.", 4,
     "VOID(EFIAPI * EFI_RESET_SYSTEM) (IN EFI_RESET_TYPE ResetType, IN "
     "EFI_STATUS ResetStatus, IN UINTN DataSize, IN VOID *ResetData OPTIONAL)",
     "ResetType     The type of reset to perform.\n\
ResetStatus   The status code for the reset.\n\
DataSize      The size, in bytes, of WatchdogData.\n\
ResetData     For a ResetType of EfiResetCold, EfiResetWarm, or EfiResetShutdown the
data buffer starts with a Null-terminated string, optionally followed by additional
binary data.",
     "IN EFI_RESET_TYPE ResetType", "IN EFI_STATUS ResetStatus", "IN UINTN DataSize",
     "IN VOID *ResetData OPTIONAL", "", "", "", "", 0},
    {"UpdateCapsule", 0x70, 0x44,
```

```
     "Passes capsules to the firmware with both virtual and physical mapping. "
     "Depending on the intended consumption, the firmware may process the "
     "capsule immediately. If the payload should persist across a system "
     "reset, the reset value returned from EFI_QueryCapsuleCapabilities must "
     "be passed into ResetSystem() and will cause the capsule to be processed "
     "by the firmware as part of the reset process.",
     3,
     "EFI_STATUS(EFIAPI * EFI_UPDATE_CAPSULE) (IN EFI_CAPSULE_HEADER "
     "**CapsuleHeaderArray, IN UINTN CapsuleCount, IN EFI_PHYSICAL_ADDRESS "
     "ScatterGatherList OPTIONAL)",
     "CapsuleHeaderArray   Virtual pointer to an array of virtual pointers to the
capsules being passed into update capsule.\n\
CapsuleCount         Number of pointers to EFI_CAPSULE_HEADER in CaspuleHeaderArray.\n\
ScatterGatherList    Physical pointer to a set of EFI_CAPSULE_BLOCK_DESCRIPTOR that
describes the location in physical memory of a set of capsules.",
     "IN EFI_CAPSULE_HEADER **CapsuleHeaderArray", "IN UINTN CapsuleCount",
     "IN EFI_PHYSICAL_ADDRESS ScatterGatherList OPTIONAL", "", "", "", "", "", 0},
    {"QueryCapsuleCapabilities", 0x78, 0x48,
     "Returns if the capsule can be supported via UpdateCapsule().", 4,
     "EFI_STATUS(EFIAPI * EFI_QUERY_CAPSULE_CAPABILITIES) (IN "
     "EFI_CAPSULE_HEADER **CapsuleHeaderArray, IN UINTN CapsuleCount, OUT "
     "UINT64 *MaximumCapsuleSize, OUT EFI_RESET_TYPE *ResetType)",
     "CapsuleHeaderArray   Virtual pointer to an array of virtual pointers to the
capsules being passed into update capsule.\n\
CapsuleCount         Number of pointers to EFI_CAPSULE_HEADER in CaspuleHeaderArray.\n\
MaxiumCapsuleSize    On output the maximum size that UpdateCapsule() can support as an
argument to UpdateCapsule() via CapsuleHeaderArray and ScatterGatherList.\n\
ResetType            Returns the type of reset required for the capsule update.",
     "IN EFI_CAPSULE_HEADER **CapsuleHeaderArray", "IN UINTN CapsuleCount",
     "OUT UINT64 *MaximumCapsuleSize", "OUT EFI_RESET_TYPE *ResetType", "", "", "", "",
     0},
    {"QueryVariableInfo", 0x80, 0x4c, "Returns information about the EFI
variables.", 4,
     "EFI_STATUS(EFIAPI * EFI_QUERY_VARIABLE_INFO) (IN UINT32 Attributes, OUT "
     "UINT64 *MaximumVariableStorageSize, OUT UINT64 "
     "*RemainingVariableStorageSize, OUT UINT64 *MaximumVariableSize)",
     "Attributes                    Attributes bitmask to specify the type of
variables on which to return information.\n\
MaximumVariableStorageSize     On output the maximum size of the storage space
available for the EFI variables associated with the attributes specified.\n\
RemainingVariableStorageSize   Returns the remaining size of the storage space
available for the EFI variables associated with the attributes specified.\n\
MaximumVariableSize            Returns the maximum size of the individual EFI variables
associated with the attributes specified.",
     "IN UINT32 Attributes", "OUT UINT64 *MaximumVariableStorageSize",
     "OUT UINT64 *RemainingVariableStorageSize", "OUT UINT64 *MaximumVariableSize", "",
     "", "", "", 0}};
}
```

The End

This PDF was generated by gitprint.me